

(19) 日本国特許庁 (J P)

(12) 公表特許公報 (A)

(11) 特許出願公表番号

特表2002-503370

(P2002-503370A)

(43) 公表日 平成14年1月29日 (2002.1.29)

(51) Int.Cl.<sup>7</sup>

G 0 6 F 9/38

識別記号

3 7 0

F I

G 0 6 F 9/38

テマコード\* (参考)

3 7 0 C

審査請求 未請求 予備審査請求 有 (全134頁)

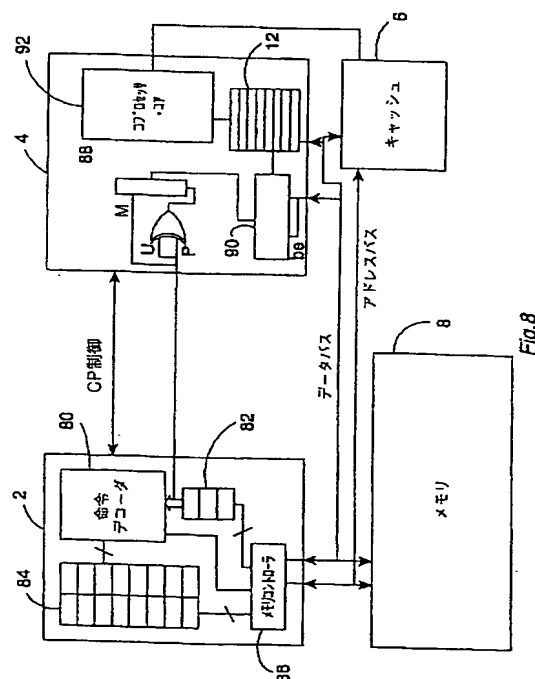
(21) 出願番号 特願平11-501806  
 (86) (22) 出願日 平成10年1月12日 (1998.1.12)  
 (85) 翻訳文提出日 平成11年11月25日 (1999.11.25)  
 (86) 国際出願番号 PCT/GB 98/00083  
 (87) 国際公開番号 WO 98/57256  
 (87) 国際公開日 平成10年12月17日 (1998.12.17)  
 (31) 優先権主張番号 9712041.4  
 (32) 優先日 平成9年6月10日 (1997.6.10)  
 (33) 優先権主張国 イギリス (GB)  
 (81) 指定国 EP (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), CN, IL, JP, KR, RU

(71) 出願人 エイアールエム リミテッド  
 イギリス国シービー1 9エヌジェイ ケンブリッジ、チェリー ヒントン、フルバーン ロード 110  
 (72) 発明者 ヨーク、リチャード  
 イギリス国 ケンブリッジ、チェリー ヒントン、パイオレット クローズ 6  
 (72) 発明者 シール、デビッド、ジェームズ  
 イギリス国 ケンブリッジ、ケインズ ロード 68  
 (72) 発明者 シムズ、ドミニク  
 イギリス国 ケンブリッジ、チェリー ヒントン、ウェッジウッド ドライブ 7  
 (74) 代理人 弁理士 浅村 皓 (外3名)

(54) 【発明の名称】 コプロセッサ・データ・アクセス制御

## (57) 【要約】

中央処理装置コア (2) と、メモリ (8) と、コプロセッサ (4) を備えたデジタル信号処理システムが、コプロセッサ・メモリアクセス命令 (例えば、LDC, STC) を使用して動作する。これらのコプロセッサ・メモリアクセス命令内のアドレッシングモード情報 (P, U, W, Offset) は、中央処理装置コア (2) によって使用されるアドレッシングモードを制御するだけでなく、コプロセッサ (4) が、何個のデータワードが転送されるのかを決めるのにも使用され、コプロセッサ (4) は、適当な瞬間に転送を終了することができる。転送されるワードの個数を事前に知っておくことは、同期DRAMと一緒に使用されるバスシステムにおいても有利である。命令内のオフセットフィールドは、特定の命令が実行されるときに、中央処理装置コア (2) によって提供される値に対する変更を指定するのに使用され、また、転送されるワード数を指定するのに使用することができる。このような装置は、デジタル信号処理動作のような規則的なデータアレイを介する動作にうまく適合する。オフセットフィールドが使用されない場



(19) 日本国特許庁 (J P)

(12) 公表特許公報 (A)

(11) 特許出願公表番号  
特表2002-503370  
(P2002-503370A)

(43) 公表日 平成14年1月29日 (2002.1.29)

(51) Int.Cl.<sup>7</sup>

G 0 6 F 9/38

識別記号

3 7 0

F I

G 0 6 F 9/38

テ-マコード (参考)

3 7 0 C

審査請求 未請求 予備審査請求 有 (全134頁)

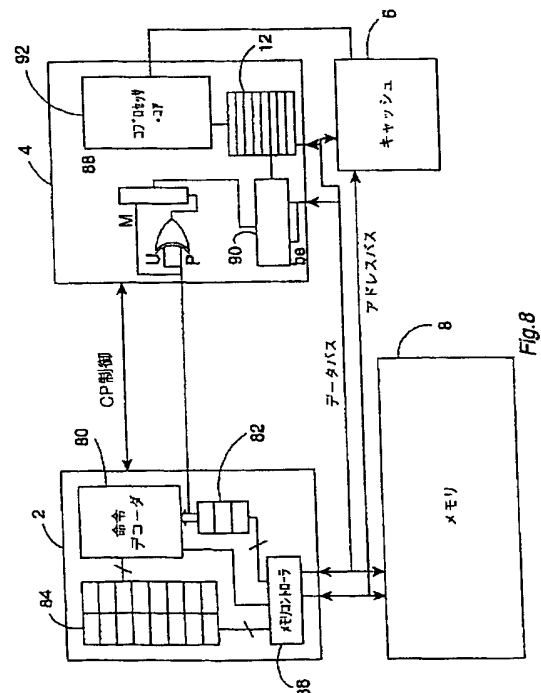
(21) 出願番号 特願平11-501806  
 (86) (22) 出願日 平成10年1月12日 (1998.1.12)  
 (85) 翻訳文提出日 平成11年11月25日 (1999.11.25)  
 (86) 国際出願番号 PCT/GB98/00083  
 (87) 国際公開番号 WO98/57256  
 (87) 国際公開日 平成10年12月17日 (1998.12.17)  
 (31) 優先権主張番号 9712041.4  
 (32) 優先日 平成9年6月10日 (1997.6.10)  
 (33) 優先権主張国 イギリス (GB)  
 (81) 指定国 EP(AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), CN, IL, JP, KR, RU

(71) 出願人 エイアールエム リミテッド  
 イギリス国シービー1 9エヌジェイ ケンブリッジ、チェリー ヒントン、フルバーン ロード 110  
 (72) 発明者 ヨーク、リチャード  
 イギリス国 ケンブリッジ、チェリー ヒントン、パイオレット クローズ 6  
 (72) 発明者 シール、デビッド、ジェームズ  
 イギリス国 ケンブリッジ、ケインズ ロード 68  
 (72) 発明者 シムズ、ドミニク  
 イギリス国 ケンブリッジ、チェリー ヒントン、ウェッジウッド ドライブ 7  
 (74) 代理人 弁理士 浅村 皓 (外3名)

(54) 【発明の名称】 コプロセッサ・データ・アクセス制御

## (57) 【要約】

中央処理装置コア (2) と、メモリ (8) と、コプロセッサ (4) を備えたデジタル信号処理システムが、コプロセッサ・メモリアクセス命令 (例えば、LDC, STC) を使用して動作する。これらのコプロセッサ・メモリアクセス命令内のアドレッシングモード情報 (P, U, W, Offset) は、中央処理装置コア (2) によって使用されるアドレッシングモードを制御するだけでなく、コプロセッサ (4) が、何個のデータワードが転送されるのかを決めるのにも使用され、コプロセッサ (4) は、適当な瞬間に転送を終了することができる。転送されるワードの個数を事前に知っておくことは、同期DRAMと一緒に使用されるバスシステムにおいても有利である。命令内のオフセットフィールドは、特定の命令が実行されるときに、中央処理装置コア (2) によって提供される値に対する変更を指定するのに使用され、また、転送されるワード数を指定するのに使用することができる。このような装置は、デジタル信号処理動作のような規則的なデータアレイを介する動作にうまく適合する。オフセットフィールドが使用されない場



## 【特許請求の範囲】

## 1. データ処理装置であって、

コプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ処理動作を行う中央処理装置と、

前記中央処理装置に結合され、データワードを保持するメモリと、

前記中央処理装置と前記メモリに結合されたコプロセッサであって、コプロセッサにより処理されるメモリ内のデータワードをアドレスするのに、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、複数のアドレッシングモードの1つを使用することを特徴とするコプロセッサと

を備えるデータ処理装置であって、

少なくとも1つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリをアクセスするのに複数のアドレッシングモードのどれを使用するかを制御するアドレッシングモード情報を含み、前記コプロセッサは、前記アドレッシングモード情報の少なくとも一部を使用して、前記少なくとも1つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御する

データ処理装置。

2. 請求項1に記載の装置であって、前記少なくとも1つのコプロセッサ・メモリアクセス命令が、アドレス値を持つ前記中央処理装置内のレジスタを参照し、前記アドレスモード情報がオフセットフィールドを含み、そこでは、アクセスされるべき前記メモリ内の開始アドレスが、前記少なくとも1つのコプロセッサ・メモリアクセス命令の実行に際して、前記アドレス値と前記オフセット値の少なくとも1つから決まることを特徴とする装置。

3. 請求項2に記載の装置であって、前記アドレス値への前記変更が、最終アドレス値を生成し、それが前記レジスタに書き戻されることを特徴とする装置。

4. 請求項2と3のいずれかに記載の装置であって、前記オフセットフィールドの少なくとも一部が、前記コプロセッサにより使用されて、前記メモリと前記コプロセッサとの間で転送されるデータの個数を制御することを特徴とする装置。

5. 請求項4に記載の装置であって、前記アドレッシングモード情報が1つ又は2つ以上のフラグを持ち、それが、前記複数のアドレッシングモードのどれが使用されているかを制御し、且つ、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを定める際に、前記オフセットフィールドが前記コプロセッサによって使用されるべきかどうかを制御することを特徴とする装置。

6. 請求項5に記載の装置であって、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを定める際に、前記オフセットフィールドが前記コプロセッサによって使用されない場合、固定数のデータワードが前記メモリと前記コプロセッサとの間で転送されることを特徴とする装置。

7. 請求項5に記載の装置であって、前記レジスタがアドレス値  $R_n$  を記憶し、データワードの長さが  $WL$  バイトであり、前記オフセット値が  $M$  であり、前記1つ又は2つ以上のフラグが、3つ以上の値ビットを備え、それらが、前記少なくとも1つのコプロセッサ・メモリアクセス命令を選択し、以下の1つに従って動作することを特徴とする装置。

	転送開始 アドレス値	アドレスレジスタ 内の最終値	転送される データワード個数
(i)	$R_n$	$R_n - (WL * M)$	( 固定値 )
(ii)	$R_n$	$R_n$	$M$
(iii)	$R_n$	$R_n + (WL * M)$	$M$
(iv)	$R_n - (WL * M)$	$R_n$	$M$
(v)	$R_n - (WL * M)$	$R_n - (WL * M)$	$M$
(vi)	$R_n + (WL * M)$	$R_n$	( 固定値 )
(vii)	$R_n + (WL * M)$	$R_n + (WL * M)$	( 固定値 )

8. 請求項7に記載の装置であって、前記フラグが、

( i ) 前記アドレス値が元々前記レジスタに記憶されていたものであるか、前記オフセットフィールドによって変更されたアドレス値であるかを指定するフラグ

ビットPと、

(i i) 前記変更が、元々レジスタに記憶されていた値から、前記オフセットフィールドで指定された値の加算又は減算のどちらであるかを指定するフラグビットUと、

(i i i) 前記アドレスレジスタ内の前記最終値が、前記レジスタに書き戻されるべきかどうかを指定するフラグビットWと

を備えていることを特徴とする装置。

9. 請求項8に記載の装置であって、前記コプロセッサがP E O R Uを求めて、データワードを1個又はM個転送すべきかを決定することを特徴とする装置。

10. 請求項8に記載の装置であって、前記レジスタが前記中央処理装置のプログラムカウンタレジスタPCであり、前記コプロセッサが、転送されるデータワードが1個かM個かを決めるために、P E O R (U O R (レジスタはPC))を求めることを特徴とする装置。

11. 以上の請求項のいずれかに記載の装置であって、前記中央処理装置及び前記コプロセッサが、デジタル信号処理を行い、前記メモリと前記コプロセッサとの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備えることを特徴とする装置。

12. 請求項6及び請求項7乃至11のいずれかに記載の装置であって、前記固定数のデータワードが単一のデータワードを含む事を特徴とする装置。

13. データ処理方法であって、

中央処理装置によりコプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ処理動作を行うステップと、

データワードを、前記中央処理装置に結合されたメモリで保持するステップと、前記中央処理装置と前記メモリに結合されたコプロセッサによって、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、複数のアドレッシングモードの1つを使用することによって、処理されるメモリ内のデータワードをアドレスするステップと

を備えるデータ処理方法であって、

少なくとも1つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリアクセスするのに複数のアドレッシングモードのどれを使用するかを制御するアドレッシングモード情報を含み、前記コプロセッサは、前記アドレッシングモード情報の少なくとも一部を使用して、前記少なくとも1つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御する

データ処理方法。

## 【発明の詳細な説明】

## コプロセッサ・データ・アクセス制御

本発明は、データ処理システムの分野に関する。特に、中央処理装置、メモリ、コプロセッサを備えたデータ処理システムに関するもので、中央処理装置とコプロセッサにより一緒に実行される命令の制御下で、データワードがメモリとコプロセッサの間でやりとりされるシステムに関する。

中央処理装置、メモリ、コプロセッサ、例えば英国ケンブリッジのアドヴァンスト (Advanced) RISC マシン株式会社によって製造された ARM マイクロプロセッサを備えたコプロセッサを備えた演算システムは公知である。そのような公知のシステムにおいて、中央処理装置は、(例えば、コプロセッサにロードする、あるいは記憶する) コプロセッサのメモリアクセス命令を実行する。これらの命令は、メモリへの供給のための適切なアドレスデータを生成し、コプロセッサがメモリと直接的にデータワード(データ転送の単位)を交換できるようにコプロセッサを準備させる。中央処理装置により開始アドレス情報がメモリに提供されると、中央処理装置を介したり、中央処理装置に記憶される、コプロセッサへ直接的にデータワードが渡されれば、最も効率が良い。コプロセッサへのそのような直接的転送を行えば、コプロセッサがデータ転送の終了を制御することによって、中央処理装置を変更する必要なしに、異なったワード数を持つ異なったコプロセッサを中央処理装置に付属させることができる。データ転送の終了を制御するためには、中央処理装置上で実行される命令に応答して、コプロセッサは、何個のデータワードが転送されるべきかを決定しなければならない。

1つの可能性として、各命令が中央処理装置上で単一のワードだけの転送を行うことができる。ただし、これはデータメモリバンド幅の使用においても、また、コード側や命令メモリバンド幅においても非常に能率が悪いので、バーストモード転送が所望される。即ち、開始アドレスがメモリに提供され、メモリが隣接する一連のメモリ位置からデータワードを返すのが好ましい。バーストモード転送の場合、効率は良くなるが、困難も生じる。即ち、何個のデータワードが

転送されるのかをコプロセッサが決定することによって、コプロセッサが転送を終了するのに必要な制御を行えるようにしなければならない。

(例えば、ARM浮動小数点アクセラレータユニットにおけるように)、中央処理装置上で実行する命令内にビットフィールドを割り当てることが公知であり、そこでは、ビットフィールド命令がコプロセッサに渡され、転送されるべきデータワードの個数をコプロセッサに指定する。しかしながら中央処理装置上で実行される命令内で使えるビットスペースには限界があり、もし、命令内のビットがコプロセッサへのデータワードの個数を渡すことに専念すると、命令内の他のフィールド用に使えるビットスペースを制限することになる。それは例えば、命令実行に続く中央処理装置内のアドレスポインタの変更のような、データ転送に関する他のパラメータを指定するのに使用されるかもしれない。

本発明の一面は、データ処理用装置であって、

コプロセッサメモリアクセス命令を含むデータ処理動作を行う中央処理装置命令を実行する中央処理装置と、

この中央処理装置に結合され、データワードを保持するメモリと、

中央処理装置とメモリに結合されたコプロセッサであって、このコプロセッサにより処理されるべきメモリ内のデータワードの指定が、中央処理装置により実行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレッシングモードの1つを使用して行われることを特徴とするコプロセッサと

を備え、

少なくとも1つのコプロセッサメモリアクセス命令が、前記複数のアドレッシングモードのどれが中央処理装置によって使用されるかを制御するアドレッシングモード情報を含み、コプロセッサは、前記アドレッシングモード情報の少なくとも一部を使用して、前記少なくとも1つのコプロセッサメモリアクセス命令に応答してメモリとコプロセッサの間で何個のデータワードが転送させるべきかを制御する。

本発明は、中央処理装置が複数のアドレッシングモードのどれを使用するかを制御するのに使用する(レジスタ番号や即値定数を含むことのある)ビットフィールドは、コプロセッサに、転送されるべきデータワードの個数を指定するのにも



使用される（命令内の他のフィールドあるいは制御レジスタに書き込まれた値のような他の因子と結合されている可能性がある）ことがある。例えば、これまでにわかっているところ、多くの場合、中央処理装置が、転送及び／又はアドレスポインタへの変更に使用されアドレスを制御するのに使用するビットフィールド情報は、コプロセッサに転送されるデータワードの個数と関係づけられるので、このビットフィールドは、コプロセッサによっても中央処理装置によっても読むことができる。コプロセッサメモリアクセス命令内で同じビットフィールドを重複して使用することにより、このようなコプロセッサメモリアクセス命令内のビットスペースを他の目的に開放することになる。更に、現在わかっているところでは、大多数の場合、転送されるべきデータワードは、少数のカテゴリに分類されるので、コプロセッサに転送されるべき個数のデータワード用に専用ビットフィールドを提供するフルフレキシビリティは、本発明を使用することによって得られるコード密度と性能の改善に負ける。

中央処理装置は、いくつかの異なる方法でアドレッシングを制御できるのに対して、本発明の好ましい実施の形態においては、前記少なくとも1つのコプロセッサメモリアクセス命令は、アドレス値を保持する中央処理装置内のレジスタを参照し、前記アドレスモード情報は、オフセットフィールドを備え、アクセスされるべきメモリ内の開始アドレスの決定は、前記少なくとも1つのコプロセッサメモリアクセス命令の実行により、前記アドレス値と前記オフセット値の少なくとも1つから行われる。

中央処理装置のレジスタ内に保持されるアドレスポインタのそのような使用は、多大なフレキシビリティを提供し、命令内のオフセットと結合されて、アプリケーションのコーディングを簡単にする。

コプロセッサとデータアレイを介して動作することが所望される場合、本発明の好ましい実施の形態では、前記アドレス値に行われた変更が、最終アドレス値を生成し、それを前記レジスタへ記憶しなおすことができるようにする。

前記オフセットフィールドの少なくとも一部が前記コプロセッサに使用されて、データワードが何個、前記メモリと前記コプロセッサの間で転送されるかを制御できることは、非常に便利であることがわかった。

このような装置は、メモリ内に保持されているデータ上にデータ処理操作をするためにコプロセッサが使用される実際の状況のかなりの部分の要求を満たすことができる。

また、アドレッシングモード情報が1つ以上のフラグを備え、前記複数のアドレッシングモードのうちのどれが使用されるか、また、何個のデータワードが前記メモリと前記コプロセッサの間で転送されるかを決定するのに前記オフセットフィールドが使用されるか否かを制御できることは有利である。

アドレッシングモード情報内の他のフラグビットに依存してオフセットフィールドを選択的に使用することによって、転送されるデータワードの個数をコプロセッサが制御できる方法のオプションの数を増加させ、それにより、コプロセッサメモリアクセス命令内のビット空間を余分に必要とすることなしに、実際の状況の、より高い部分の要求に合わせることができる。

高い割合の所望のタイプの動作を成就するには、好ましい実施の形態は、次のようなものでなければならない。即ち、前記コプロセッサが、前記オフセットを使用せずに、前記メモリと前記コプロセッサの間で転送されたデータワードの個数を決定する場合、メモリとコプロセッサの間では、固定数のワードが転送される。

コプロセッサを制御するコプロセッサメモリアクセス命令モードの相補セットは、次のようなものである。即ち、前記レジスタがアドレス  $R_n$  を記憶し、データワードが  $WL$  バイトの長さで、前記オフセット値が  $M$  であり、前記1つ以上のフラグが、3つ以上の値ビットを備え、それらが、前記少なくとも1つのコプロセッサアクセス命令を選択し、以下の1つに従って操作する。

	転送開始 アドレス値	アドレスレジスタ 内の最終値	転送される ワード個数
(i)	$R_n$	$R_n - (WL * M)$	( 固定値 )
(ii)	$R_n$	$R_n$	$M$
(iii)	$R_n$	$R_n + (WL * M)$	$M$
(iv)	$R_n - (WL * M)$	$R_n$	$M$
(v)	$R_n - (WL * M)$	$R_n - (WL * M)$	$M$
(vi)	$R_n + (WL * M)$	$R_n$	( 固定値 )
(vii)	$R_n + (WL * M)$	$R_n + (WL * M)$	( 固定値 )

コプロセッサメモリアクセス命令は、以下のようなフラグを備えると有利である。

( i ) 前記開始アドレス値が、元々前記レジスタに記憶されていたアドレス値であるか、前記オフセットフィールドによって指定される変更されたアドレス値であるかを指定するフラグビット P、

( i i ) 前記変更が、前記レジスタに元々記憶されていた値からオフセットフィールドによって指定される値を加算したものか減算したものであるかを指定するフラグビット U、

( i i i ) 前記アドレスレジスタ内の前記最終値が、前記レジスタに記憶されなすべきか否かを指定するフラグビット W。

このようなフラグセットを使用してコプロセッサは、高速で簡単な動作制御を行うことのできる構成となる。即ち、 $P \text{ EOR } U$ を求めることによって、1個又はM個のデータワードが転送されるべきかを決定することができる。

更に、ベースレジスタがARMプログラムカウンタレジスタ ( PC又はR15 ) であれば、転送されるワード数1を生成することができる。この場合、単一ワード転送を決定するロジックが、 $P \text{ EOR } (U \text{ 又は } ( \text{ベースレジスタがPC} ) )$ に変更される。

前述のコプロセッサと、中央処理装置及びメモリとの相互作用を制御する特性は多数の異なるフィールドに使用できる（例えば、浮動小数点コプロセッサ）場合、データアクセスは、次のように、比較的規則的である。即ち、前記中央処理装置と前記コプロセッサがデジタル信号処理を行い、前記メモリと前記コプロセッサの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備える。

本発明を他の面から見ると、本発明は以下のステップを備えるデータ処理方法を提供する。即ち、

中央処理装置により、コプロセッサメモリアクセス命令を含む中央処理装置命令を実行してデータ処理操作を行うステップと、

前記中央処理装置に結合されたメモリにおいてデータワードを保持するステップと、

前記中央処理装置に結合されたコプロセッサにより処理される前記メモリ内のデータワードをアドレスするステップであって、前記メモリは、前記中央処理装置により実行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレッシングモードの1つを使用するステップと

を備え、

前記コプロセッサメモリアクセス命令の少なくとも1つは、前記複数のアドレッシングモードのどれを中央処理装置が使用して前記メモリをアクセスするかを制御するアドレッシングモード情報を含み、前記コプロセッサは、前記少なくとも1つのコプロセッサメモリアクセス命令に応答して、前記アドレッシングモード情報の少なくとも一部を使用して何個のデータワードが前記メモリと前記コプロセッサの間で転送されるかを制御する方法である。

本発明の実施の形態について、以下に添付図面を参照しながら、例を示す。

図1は、デジタル信号処理装置のハイレベルの構成を示し、

図2は、コプロセッサの入力バッファとレジスタ構成を示し、

図3は、コプロセッサ内のデータパス（datapath）を示し、

図4は、レジスタからハイ又はローのビットを読むためのマルチプレクシング回路を示し、

図5は、好ましい実施の形態におけるコプロセッサにより使用されるレジスタ・リマッピング・ロジックを示すブロック図であり、

図6は、図5に示されたレジスタ・リマッピング・ロジックを更に詳しく示し、

図7は、ブロック・フィルタ・アルゴリズムを示す表であり、

図8は、中央処理装置と、メモリと、コプロセッサアクセスメモリ命令を実行するためのコプロセッサとを備えたシステムを模式的に示し、

図9は、図8のシステムの動作に対応するフロー・ダイアグラムである。

以下の説明において、セクション1では、中央処理装置と、メモリと、高速デジタル信号処理能力を持つコプロセッサとを備えたシステムについて説明する。セクション2は、セクション1のシステムの変形について述べるもので、ここでは、コプロセッサメモリアクセス命令が、転送されるデータワードの個数のコプロセッサによる制御をより簡単にすべく変更されている。

#### セクション1

以下に説明するシステムは、デジタル信号処理(DSP)に関する。DSPは、いろいろな形態を取ることができるが、典型的には、大量のデータの高速(実時間)処理を必要とする処理である。このデータは、典型的には、アナログの物理的信号である。DSPの好例として、デジタル移動電話に使用されるものがある。そこでは、無線信号が送受信され、アナログ音声信号から、及びアナログ音声信号へのデコーディング及びエンコーディング(典型的には、畳み込み(convolution)、変換、相関の操作を使用)が必要となる。また、他の例として、ディスクヘッドからの信号が処理されてヘッド・トラッキング制御が行われるディスク・ドライバ・コントローラが挙げられる。

上記のような文脈において、マイクロプロセッサ・コア(ここでは、英国、ケンブリッジのアドヴァンスト・RISC・マシン・リミテッドにより設計されたマイクロプロセッサの範囲からのARMコア)上でのデジタル信号処理システムの説明をする。マイクロプロセッサとコプロセッサ・アーキテクチャとの間のインターフェースは、それ自体が、DSP機能を提供すべく具体的構成を持つ。以下の説明において、マイクロプロセッサ・コアはARM、コプロセッサはピッコ

ロ (Piccolo) とする。ARM とピッコロは、典型的には、他の構成要素（たとえば、チップ上の DRAM、ROM、D/A コンバータ、A/D コンバータ）を ASIC の部分として含む単一の集積回路として製造される。

ピッコロは、ARM のコプロセッサであるから、ARM 命令の集合の一部を実行する。ARM コプロセッサ命令により、(Load Coprocessor, LDC and Store Coprocessor, STC 命令を使用して) ARM がピッコロとメモリーの間でデータをやり取りさせ、また、(move to coprocessor, MCR, 及び、move from coprocessor, MCR 命令を使用して) ARM が ARM レジスタをピッコロとやり取りすることができる。ある見方をすれば、ARM とピッコロの相互作用は、ARM GA ピッコロのデータに対して強力なアドレス生成器として作用し、ピッコロの方は、大量のデータを実時間で扱う必要のある DSP 操作を自由に行うことによって、対応の実時間結果を生み出すことである。

図 1 は、ARM 2 がピッコロ 4 に制御信号を発行して、データワードをピッコロ 4 に対して送信させ、またデータワードをピッコロ 4 から転送させる様子を示す。命令キャッシュ 6 は、ピッコロ 4 にとって必要なピッコロプログラム命令ワードを記憶する。単一の DRAM メモリ 8 は、ARM 2 とピッコロ 4 の両方にとって必要なすべてのデータ及び命令ワードを記憶する。ARM 2 は、メモリ 8 へのアドレッシング (addressing) 及びすべてのデータ転送の制御に責任がある。単一のメモリ 8、及び 1 セットのデータバスとアドレスバスから成る構成は、複数のメモリと高い帯域幅のバスを必要とする典型的 DSP アプローチに比較して、構成が簡単であり、費用も易い。

ピッコロは、命令キャッシュ 6 からの第 2 の命令ストリーム (デジタル信号処理プログラム命令ワード) を実行し、これにより、ピッコロのデータバスが制御される。これらの命令は、デジタル信号処理方式操作、例えば、Multiply-Accumulate (乗算-累算)、及び制御フロー命令、例えば、ゼロ・オーバーヘッド・ループ命令を含む。これらの命令は、ピッコロのレジスタ 10 (図 2 を参照) に保持されているデータを操作する。このデータは、前もって、ARM 2 によってメモリ 8 から転送されたものである。複数の命令が命令キャッシュ 6 からストリ

ームとして出され、命令キャッシュ6が、データバスを、完全な支配下に置く。

小型ピッコロ命令キャッシュ6は、1行当たり16ワードの4行で、直接

マップされたキャッシュ(64個の命令)となる。導入の方法によっては、命令キャッシュをもっと大きくしてもよい。

このように、2つのタスクが独立的に走る。ARMがデータをロードして、ピッコロがそれを処理する。これにより、16ビット・データ上で単一サイクル・データ処理が維持される。ピッコロの持つデータ入力メカニズム(図2に示される)により、ARMは、シーケンシャル・データを先に取り込み、そのデータがピッコロに必要なより先にロードする。ピッコロは、ロードされたデータにどのような順序でもアクセスすることができ、古いデータが最後に使用されると、自動的にそのレジスタを再び満たす(すべての命令はソースオペランド1につき、ソースレジスタを再充填すべきであることを示す1ビットを持つ)。この入力メカニズムは、リオーダー(reorder)バッファと呼ばれ、入力バッファ12を備える。ピッコロにロードされる(以下に示すLDCまたはMCRを介して)すべての値には、その値の目的地がどのレジスタであるかを示すタグR<sub>n</sub>が付いている。タグR<sub>n</sub>は、入力バッファ内のデータワードの側に記憶される。あるレジスタがレジスタ選択回路14を介してアクセスされ、命令がデータレジスタの再充填を指定すると、そのレジスタは、信号Eによって「空き」の印がつく。すると、レジスタは、自動的に、再充填制御回路16によって、その入力バッファ12内でそのレジスタに向けられた最も早くロードされた最古の値を充填される。リオーダー・バッファは8つのタグ付き値を保持する。入力バッファ12の形式は、FIFOと似ているが、キーの中央からデータワードを抽出することができ、その後で、遅くに記憶されたワードが渡され、その空き場所を埋める。従って、入力から最も遠いデータワードが最古であり、入力バッファ12が正しいタグR<sub>n</sub>を持つ2つのデータワードを保持する時は、その最古のデータワードを使用して、どちらのデータワードでレジスタを再充填すべきかを決定することができる。

ピッコロは、図3に示されたように、データを出力バッファ18(FIFO)

に記憶させて出力する。データはF I F Oにシーケンシャルに書き込まれ、A R Mによって同じ順序でメモリ8に読み出される。出力バッファ18は、8つの32ビットの値を保持する。

ピッコロは、コプロセッサ・インターフェース（図1のC P制御信号）を介してA R Mと接続する。A R Mコプロセッサ命令の実行に際して、ピッコロは、それを実行するか、あるいは、ピッコロがその命令を実行できるようになるまでA R Mを待たせるか、あるいは命令実行を拒否することができる。最後の場合、A R Mは、未定義命令例外とする。

ピッコロが実行する最も普通のコプロセッサ命令はL D CとS T Cであり、これらは、それぞれデータワードをデータバスを介してメモリ8へ、及びメモリ8からロードし（L D C）、記憶させ（S T C）、A R Mがすべてのアドレスを生成する。リオーダ・バッファにデータをロードし、出力バッファ18からのデータを記憶するのもこれらの命令である。ピッコロは、入力リオーダ・バッファにデータをロードするのに十分な場所がなければA R MをL D Cのままにし、また出力バッファに記憶すべき十分なデータがなければA R MをS T Cのままにする。ピッコロは、また、A R M／コプロセッサ・レジスタ転送を行って、A R Mがピッコロの特定の(special)レジスタにアクセスできるようにする。

ピッコロは、それ自身の命令はメモリから取り込み、図3に示されたピッコロのデータバスを制御し、リオーダ・バッファからレジスタへ、またレジスタから出力バッファ18へデータを転送する。これらの命令を行うピッコロの演算ユニットは、乗算／加算回路20を有し、これが乗算、加算、減算、乗算・累算、論理操作、シフト、及び回転を行う。また、データバスには累算／退出(decumulate)回路22と、縮尺(scale)／飽和(saturate)回路24とが備わっている。

ピッコロ命令は、最初にメモリから命令キャッシュ6にロードされ、そこへピッコロがアクセスし、主記憶にアクセスバックする必要がない。

メモリがアボート(abort)した場合、ピッコロはそれを修復することができない。従って、ピッコロを仮想メモリシステムで使用する場合、すべてのピッコロのデータは、ピッコロのタスクの始めから終わりまで、物理的メモリになければ



ならない。このことは、ピッコロのタスクの実時間性、例えば実時間DSPを考えると、大した問題ではない。メモリ・アボートが起きると、ピッコロは停止して状態レジスタS2にフラグをセットする。

図3は、ピッコロの全体のデータパス機能を示す。レジスタ・バンク10は、3つの読み出しポートと2つの書き込みポートを使用する。1つの書き込みポート(Lポート)は、リオーダ・レジスタからレジスタを再充填するのに使用される。出力バッファ18は、ALU結果バス26から直接的に更新され、出力バッファ18からの出力は、ARMプログラム制御の支配下にある。ARMコプロセッサ・インターフェースは、LDC (Load coprocessor) 命令をリオーダ・バッファに行い、出力バッファ18からSTC (Store Coprocessor) 命令を行い、また、レジスタバンク10上にMCRとMRC (Move ARM register to/from CP register)を行う。

残りのレジスタ・ポートは、ALUに使用される。読み出しポート(A及びB)は、入力を乗算/加算回路20に駆動し、C読み出しポートは、累算 (accumulate) / 退出 (decumulate) 回路22入力の駆動に使用される。残りの書き込みポートWは、結果をレジスタバンク10に戻すのに使用される。

乗算器20は、符号付き又は符号無し16 x 16の乗算を行い、必要により48ビット累算を伴うこともできる。スケーラー (scaler) ユニット24は、0から31までの即値算術又は論理シフト右を提供することができ、その後、必要により飽和を行うことができる。シフタ (shifter) 及び論理ユニット20は、各周期でシフト又は論理操作を行うことができる。

ピッコロは、D0-D15又はA0-A3, X0-X3, Y0-Y3, Z0-Z3という名のついた16個の汎用レジスタを持つ。最初の4つのレジスタ (A0-A3) は、累算用で、48ビットの幅があり、余分な16ビットが、多数の連続的計算の間にオーバーフローが生じないためのガードを提供する。残りのレジスタは32ビットの幅である。

ピッコロのレジスタは各々2つの独立した16ビットの値を含むものとして扱うことができる。ビット0からビット15までが下半分、ビット16からビット

31までが上半分を含む。命令は、ソースオペランドとして各レジスタのどちらかの半分の16ビットを指定することができ、あるいは、全体の32ビットレジスタを指定することもできる。

また、ピッコロは、飽和演算に対する備えもある。乗算、加算、減算命令の変量は、結果が目的レジスタのサイズより大きい場合、飽和結果を提供する。目的

レジスタが48ビットのアキュムレータであれば、値は32ビットで飽和される（つまり、48ビットの値を飽和させる方法はない）。48ビットのレジスタにはオーバーフローの検出がない。これは手頃な制限である。というのは、オーバーフローを起こすには、少なくとも65536乗算累算命令が必要であるから。

各ピッコロのレジスタは、「空き」（Eフラグ、図2参照）であるか、1つの値を含む（レジスタの半分だけが空きになることはない）。初期状態では、すべてのレジスタが空きの印がついている。各周期で、ピッコロは再充填制御回路16によって、空きレジスタの1つを、入力リオーダ・バッファからの値で埋める。あるいは、レジスタにALUからの値が書き込まれている場合は、「空き」ではない。もし、レジスタにALUからの書き込みがあり、これと同時に、リオーダ・バッファからのレジスタに入れられる値が控えている場合は、結果は未定義である。空きレジスタに読み出しが行われれば、ピッコロの実行ユニットはとまってしまう。

入力リオーダ・バッファ（ROB）は、コプロセッサ・インターフェースとピッコロのレジスタ・バンクとの間にある。データがROBにロードされる時は、ARMコプロセッサが転送する。ROBは、多数の32ビットの値を含み、それぞれ値の目的地となるピッコロ・レジスタを示すタグを持っている。タグは、また、そのデータが32ビットレジスタの全体に転送されるのか、あるいは32ビット中の下の16ビットだけに転送されるべきかも示す。データがレジスタ全体に転送される場合は、そのエントリーの下の16ビットは目的レジスタの下半分に転送され、上の16ビットはレジスタの上半分に転送される（目的レジスタが48ビット・アキュムレータの場合は、符号が拡張される）。データの目的地がレジスタの下半分だけ（いわゆるハーフ・レジスタ）の場合、下の16ビットが

先に転送される。

レジスタのタグは常に物理的目的レジスタを示し、レジスタのリマッピングが行われることはない（レジスタのリマッピングについては、以下を参照）。

各周期で、ピッコロは、次のように、データ・エントリをR O Bからレジスタ・バンクへ転送しようとする。

ー R O B の各エントリが検査され、タグが空きレジスタと比較され、エントリ

の一部又は全部からレジスタへ転送が可能かどうか決定される。

ー 転送可能なエントリの集合から、最古のエントリが選択され、そのデータがレジスタバンクへ転送される。

ー このエントリのタグが更新されてエントリを空きにする。エントリの一部だけが転送された場合は、転送された部分だけが空きの印になる。

例えば、目的レジスタが完全に空きであり、選択されたR O Bエントリが含むデータが1つの全体レジスタ用であれば、32ビット全体が転送され、そのエントリは空きの印になる。目的レジスタの下半分が空きであり、R O Bの含むデータがレジスタの下半分用であれば、R O Bエントリの下の16ビットが目的レジスタの下半分へ転送され、R O Bの下半分が空きの印になる。

どのエントリでも、上の16ビットと下の16ビットは、それぞれ独立に転送することができる。レジスタバンクに転送できるデータを含むエントリが皆無の場合、その周期では、転送は行われない。下の表は、目的R O Bエントリと目的レジスタ状態のあらゆる可能な組み合わせを示す。

	目的, Rn, 状態		
目的 ROB エントリ状態	空 き	下半分 空 き	上半分 空 き
レジスタ全体 両半分有効	Rn.h <- entry.h  Rn.l <- entry.l エントリ「空 き」	Rn.l <- entry.l  エントリ 下半分 「空 き」	Rn.h <- entry.h  エントリ 上半分 「空 き」
レジスタ全体 上半分有効	Rn.h <- entry.h エントリ「空 き」		Rn.h <- entry.h エントリ「空 き」
レジスタ全体 下半分有効	Rn.l <- entry.l エントリ「空 き」	Rn.l <- entry.l エントリ「空 き」	
レジスタ半分 両半分有効	Rn.l <- entry.l エントリ 下半分 「空 き」	Rn.l <- entry.l エントリ 下半分 「空 き」	
レジスタ半分 上半分有効	Rn.l <- entry.h エントリ「空 き」	Rn.l <- entry.h エントリ「空 き」	

以上をまとめると、1つのレジスタの2つの半分は、互いに独立に、ROBから充填することができる。ROB内のデータは、レジスタ全体用に印が付けられるか、あるいはレジスタの下半分用の2つの16ビットの値としての印が付く。

データをROBにロードするにはARMコプロセッサ命令が使用される。ROBにおいてデータが印が付けられる方法は、転送に使用されたARMコプロセッサ命令がどれであったかによる。ROBにデータを充填するのに使用できるARM命令には以下のものがある。

```

L D P {<cond>} <16/32>                <dest>, [Rn] { ! } , #<size>
L D P {<cond>} <16/32>W                <dest>, <wrap>, [Rn] { ! } , #<size>
L D P {<cond>} 16U                      <bank>, [Rn] { ! }
M P R {<cond>}                          <dest>, Rn
M R P {<cond>}                          <dest>, Rn

```

R O B の構成には、以下の A R M 命令が提供される。

L D P A <bank list>

最初の 3 つは、L D C 命令としてアセンブルされ、M P R と M R P は、M C R 命令として、L D P A は C D P 命令としてアセンブルされる。

上記 <dest> は、ピッコロのレジスタ (A 0 - Z 3) を示し、R n は A R M レジスタを示し、<size> は 4 の乗数 (ゼロを除く) となる定数としてのバイト数であり、<wrap> は、定数 (1、2、4、8) を示す。{} によって囲まれた領域は、オプションである。転送がリオダ・バッファへ当てはまるようにするために、<size> は最大で 3 2 である。多くの場合、<size> は、この制限より小さくしてデッドロックを避ける。<16 / 32> 領域は、ロードされるデータが 16 ビット・データとして扱われ、endianess 特定動作 (以下を参照) を行うべきか、あるいは 32 ビットデータであるかを示す。

注 1 : 以下の説明において、L D P または L D P W に言及する場合、これらの命令の 16 ビット用変種と 32 ビット用変種の両方を含むものとする。

注 2 : 1 つのワード (word) は、メモリからの 32 ビットの固まりであり、それは、16 ビットのデータ項目 2 つから成るか、あるいは 32 ビットのデータ項目 1 つからなる。

L D P 命令は、多数のデータ項目をフル・レジスタ用として転送する。この命令は、メモリ内のアドレス R n から <size> / 4 ワードをロードし、それらを R O B に挿入する。転送することのできるワード数は以下のように制限される。

- <size> の量は、4 の非ゼロ倍数でなければならない、
- <size> は、特定の導入について R O B のサイズ以下でなければならない (第 1 版では 8 ワード、その後の版では、それ以下にならない保証があること)。

転送される最初のデータ項目は目的地が <dest> のタグを付け、第 2 のデータ

項目は、 $\langle \text{dest} \rangle + 1$  というようになる (Z 3 から A 0 まではラッピング (wrap ping) )。もし “!” が指定された場合は、その後、レジスタ R n が  $\langle \text{size} \rangle$  によって 1 つずつ増加される。

L D P 1 6 の変種が使用された場合は、エンダイアン (endian) 特定動作が 2 つの 1 6 ビットのハーフワードに行われて、それらがメモリシステムから戻される時には 3 2 ビットデータ項目とする。より詳しくは、以下の Big Endian 及び Little Endian サポートを参照せよ。

L D P W 命令は、多数のデータ項目をレジスタのセットに転送する。最初に転送されるデータ項目には  $\langle \text{dest} \rangle$  のタグが付き、次は  $\langle \text{dest} \rangle + 1$  のタグが付き、以下同様。 $\langle \text{wrap} \rangle$  転送が起きると、次に転送される項目は、 $\langle \text{dest} \rangle$  用となり、以下同様。 $\langle \text{wrap} \rangle$  の量は、ハーフワードの量で指定される。

L D P W には、次の制限がある。

- $\langle \text{size} \rangle$  の量は、4 の非ゼロ倍でなければならない、
- $\langle \text{size} \rangle$  は、特定の導入について R O B のサイズ以下でなければならない (第 1 版では 8 ワード、その後の版では、それ以下にならない保証がある)、
- $\langle \text{dest} \rangle$  は、{ A 0 , X 0 , Y 0 , Z 0 } のいずれか 1 つでよく、
- $\langle \text{wrap} \rangle$  は、L D P 3 2 W については { 2 , 4 , 8 } のいずれかの個数のハーフワードであり、L D P 1 6 W については { 1 , 2 , 4 , 8 } のいずれかの個数のハーフワードであり、
- $\langle \text{size} \rangle$  の量は、 $2 * \langle \text{wrap} \rangle$  より大きくなければならない。さもないと、ラッピングは起きず、代わりに L D P 命令が使用される。

たとえば、次の命令

```
L D P 3 2 W      X 0 , 2 , [ R 0 ] ! , # 8
```

は、2 つのワードを R O B にロードし、その目的地をフル・レジスタ X 0 とする。R 0 は、8 増加する。次の命令

```
L D P 3 2 w      X 0 , 4 , [ R 0 ] , # 1 6
```

は、4 つのワードを R O B にロードし、それらの目的地を X 0 , X 1 , X 0 , X 1 (この順序で) とする。R 0 は影響されない。

L S P 1 6 W に対しては、 $\langle \text{wrap} \rangle$  は、1、2、4、又は 8 として指定できる

1 のラップが指定されると、すべてのデータのタグの目的地が、目的レジスタの下半分 <dest>. 1. となる。これは、ハーフ・レジスタの場合である。

例えば、次の命令

```
L D P 1 6 W          X 0, 1, [R 0]!, # 8
```

は、2つのワードをR O Bにロードし、それらを16ビットデータとして目的地をX 0. 1とする。R 0は8増加される。次の命令

```
L D P 1 6 W          X 0, 4, [R 0], # 1 6
```

は、L D P 3 2 Wの例と同様に挙動するが、ただし、エンダイアン特定動作は、メモリから戻されるデータ上に行われる。

L D P 命令のすべての使用されないエンコーディングは、将来の拡張用にとっておくことができる。

L D P 1 6 U 命令は、非ワード揃え (non-word aligned) 16ビットデータの効率良い転送をサポートする。L D P 1 6 U サポートはレジスタD 4 - D 1 5 (X, Y, Zバンク) になされる。L D P 1 6 U サポートは、レジスタ32ビットワードのデータ1つ (2つの16ビットデータ項目を含む) をメモリからピッコロへ転送することになる。ピッコロは、このデータの下16ビットを捨て、上の16ビットを保持レジスタに記憶する。X, Y, Zバンク用の保持レジスタがある。バンクの保持レジスタが通報されると (primed) と、データの目的地がそのバンク内のレジスタであれば、L D P {w} 命令の挙動が変化する。R O Bにロードされたデータは、L D P 命令によって転送されつつあるデータの下16ビットと保持レジスタとの連結によって形成される。転送されつつある上の16ビットは、保持レジスタに入れられる。

```
entry <- data.1; holding__register
```

```
holding__register<-data.h
```

このモードの動作は、L D P A 命令によって打ち切られるまで続く。保持レジスタは、目的レジスタのタグもサイズも記録しない。これらの性質は、次のdata. 1. の値を提供する命令から得られる。

メモリシステムから戻されたデータには、常にエンダイアン特定挙動が起きる可能性がある。LDP16Uと同等の非16ビットはない。というのは、32ビ

ットデータ項目はすべてメモリにおいてワード揃えされるからである。

LDP A 命令は、LDP16U 命令によって開始された操作の非整列(unaligned)モードを取り止めるのに使用される。非整列モードは、バンク X, Y, Z 上で独立に切ることができる。例えば、次の命令

LDP A { X, Y }

は、バンク X, Y 上で非整列モードを打ち切る。これらのバンクの保持レジスタ内のデータは、捨てられる。

非整列モードにないバンク上でLDP Aを実行することは可能であり、そのバンクは整列モードのままである。

MPR 命令は、ARMレジスタR<sub>n</sub>の内容をROBに入れ、ピッコロレジスタ<dest>に向けられる。目的レジスタ<dest>は、A0-Z3の範囲のフルレジスタならどれでもよい。例えば、次の命令

MPR X0, R3

は、R3の内容をROBに移し、そのデータをフルレジスタX0用とする。

データがARMからピッコロに転送される時にエンダイアネス(endianess)特定挙動が生じることがない。というのは、ARMは、内部的に、あまりエンダイアンではないからである。

MPRW 命令は、ARMレジスタR<sub>n</sub>の内容をROBに入れ、それを、16ビットピッコロレジスタ<dest>. 1. 向けの2つの16ビットデータ項目とする。<dest>についての制限は、LDPW 命令の場合と同じである(つまり、Z0, X0, Y0, Z0)。例えば、次の命令

MPRW X0, R3

は、R3の内容をROBに移し、X0. 1. 向けの2つの16ビット量のデータとする。尚、1でラップするLDP16Wの場合、32ビットレジスタの下半分だけが目的地となり得る。

MPR については、データに対してエンダイアネス特定操作は何も行われない



LDPは、次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				110			P	U	N	W	I	Rn					DEST			PICCOLO1				SIZE/4							

ここで、PICCOLO1は、ピッコロの最初のコプロセッサの番号（現在8）である。NビットがLDP32（1）とLDP16（0）との間の選択を行う。

LDPWは、次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				110			P	U	N	W	I	Rn					DES	WRA	PICCOLO2				SIZE/4								

ここで、DESTは、目的レジスタA0，X0，Y0，Z0に対する0－3であり、WRAPは、1、2、4、8の値のラップに対して0－3である。PICCOLO2は、ピッコロの第2のコプロセッサ番号（現在9）である。Nビットが、LDP32（1）とLDP16（0）との間の選択を行う。

LDP16Uは、次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				110			P	U	0	W	I	Rn					DES	01	PICCOLO2				00000001								

ここで、DESTは、目的バンクX，Y，Zに対する1－3である。

LDP Aは、次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND				1110			0000			0000			0000			PICCOLO1				000		0	BANK								

ここで、BANK [3 : 0] は、バンクごとの非整列モードを打ち切るのに使用

される。BANK [ 1 ] がセットされると、バンク X 上の非整列モードが打ち切られる。BANK [ 2 ] 及びBANK [ 3 ] がセットされれば、それぞれバンク

Y, Z 上の非整列モードが打ち切られる。尚、これはCDP操作である。

MPRは、次のようにエンコードされる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	1110	0	1	0	0	DEST	Rn	PICCOLO1	000	1	0000
------	------	---	---	---	---	------	----	----------	-----	---	------

MPRWは、次のようにエンコードされる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	1110	0	1	0	0	DEST	00	Rn	PICCOLO2	000	1	0000
------	------	---	---	---	---	------	----	----	----------	-----	---	------

ここで、DESTは、目的レジスタX0, Y0, Z0に対する1-3である。

出力FIFOは、32ビットの値を8つまで保持することができる。これらは、次の(ARM)オペコード(opcodes)の1つを使用して、ピッコロから転送される。

STP {<cond>} <16/32> [Rn] {!}, #<size>

MRP Rn

最初のは、<size>/4ワードを出力FIFOから、ARMレジスタRnによって与えられるアドレスへ退避する。“!”があれば、Rnを指示する。デッドロックを避けるために、<size>は、出力FIFOのサイズ(この導入例では8エントリ)以下でなければならない。STP16の変種が使用された場合は、メモリシステムから戻されるデータにエンディアン特定挙動が生じる可能性がある。

MRP命令は、出力FIFOから1つのワードを除去し、それをARMレジスタRnに入れる。MRPと同様に、このデータには、エンディアン特定操作が適用されることはない。

S T P 用の A R M エンコーディングは以下の通り。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	110	P	U	N	W	0	Rn	0000	PICCOLO1	SIZE/4
------	-----	---	---	---	---	---	----	------	----------	--------

ここで、Nは、S T P 3 2 ( 1 ) と S T P 1 6 ( 0 ) との間の選択を行う。P, U, W ビットの定義については、A R M データシートを参照せよ。

M R P 用の A R M エンコーディングは以下の通り。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	1110	0	1	0	1	0000	Rn	PICCOLO1	000	1	0000
------	------	---	---	---	---	------	----	----------	-----	---	------

ピッコロ命令セットは、内部的にはエンダイアン操作がほとんどないと仮定している。例えば、32ビットレジスタに、複数の16ビット・ハーフとしてアクセスする場合、下半分がビット15から0を占めるとする。ピッコロは、大きなエンダイアン・メモリ又は周辺機器(peripherals)のあるシステムで動作することになるので、16ビットでパックされたデータを正しくロードできるようにしなければならない。

ピッコロ(つまり、D S P が採用されたコプロセッサ)は、A R M (例えば、英国、ケンブリッジのアドヴァンスト R I S C マシンス・リミテッドによって製造された A R M 7 マイクロプロセッサ)のように、プログラマがプログラム可能周辺機器で制御できるであろう ' B I G E N D ' 構成ピンを持っている。ピッコロは、このピンを使用して入力リオーダ・バッファ及び出力 F I F O を構成する。

A R M がパック16ビットデータをリオーダ(reorder)・バッファにロードする時は、そのことを、L D P 命令の16ビット形式を使用して示さなければならない。この情報は ' B I G E N D ' 構成入力の状態と組み合わせられて、データを保持ラッチへ入れ且つリオーダ・バッファを適当な順序にする。特にbig endian モードの時は、保持レジスタはロードされたワードの下16ビットを記憶し、次のロードの上16ビットと対(ペア)にされる。保持レジスタの内容は、常に、

リオーダ・バッファへ転送されたワードの下16ビットで終わる。

出力FIFOは、パックされた16ビット又は32ビットデータを含むことができる。プログラマは、STP命令の正確な形式を使用して、16ビットデータがデータバスの正しい半分に提供されていることをピッコロが確認できるようにしなければならない。big endianとして構成されている場合、STPの16ビッ

ト形式が使用されると、上16ビットハーフ及び下16ビットハーフが交換される。

ピッコロは、ARMからしかアクセスできないプライベート・レジスタを4つ持っている。これらは、S0-S2と呼ばれる。これらにアクセスできるのは、MRC命令とMCR命令だけである。オペコードは以下の通り。

MPSR            S<sub>n</sub>, R<sub>m</sub>

MRPS            R<sub>m</sub>, S<sub>n</sub>

これらのオペコードは、ARMレジスタR<sub>m</sub>とプライベート・レジスタS<sub>n</sub>との間で32ビット値を転送する。それらは、ARMにおいて、コプロセッサ・レジスタ転送としてエンコードされる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	1110	001	L	S <sub>n</sub>	R <sub>m</sub>	PICCOLO	000	I	0000
------	------	-----	---	----------------	----------------	---------	-----	---	------

ここで、Lは、MPSRなら0、MRPSなら1である。

レジスタS0は、ピッコロの一意なID及び改定コードを含む。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Implementor	Architecture	Part Number	Revision
-------------	--------------	-------------	----------

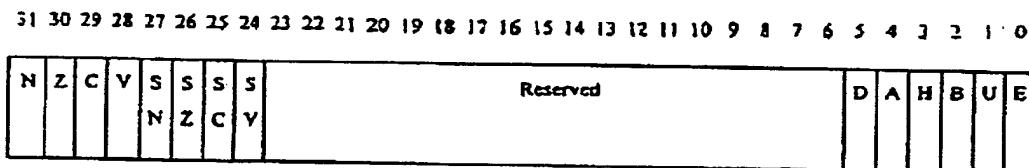
[3:0]ビットは、プロセッサの改定番号を含む。

[15:4]ビットは、2進符号化された10進フォーマットの3桁部分の番号(ピッコロなら、0x500)を含む。

[ 2 3 : 1 6 ] ビットは、アーキテクチャ版数を含む。0 x 0 0 = 第 1 版

[ 3 1 : 2 4 ] ビットは、導入者の商標の A S C I I コードを含む。0 x 4 1  
= A = A R M L t d .

レジスタ S 1 は、ピッコロの状態レジスタである。



一次状態コードフラグ ( N , Z , C , V )

二次状態コードフラグ ( S N , S Z , S C , S V )

E ビット : ピッコロは、A R M によってディスエーブルされ、中止した。

U ビット : ピッコロは、U N D E F I N E D ( 未定義 ) 命令に出会って、中止した。

B ビット : ピッコロは、B R E A K P O I N T ( 区切点 ) に出会って、中止した。

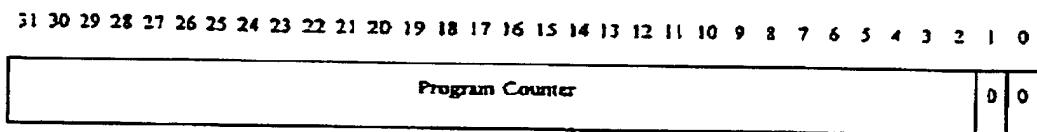
H ビット : ピッコロは、H A L T ( 中止 ) 命令に出会って、中止した。

A ビット : ピッコロは、メモリ・アボート ( ロード、ストア、又はピッコロ命令 ) によって、中止した。

D ビット : ピッコロは、デッドロック条件を検出し、中止した ( 以下を参照 )

。

レジスタ S 2 はピッコロプログラム・カウンタである。



プログラム・カウンタに書き込みすると、ピッコロはそのアドレスで ( 中止状態であれば中止状態のまま ) プログラムの実行を始める。プログラム・カウンタはリセットされた時、未定義である。というのは、プログラム・カウンタへの書

き込みによつて、ピッコロは常にスタートされるからである。

実行中、ピッコロは命令の実行及びコプロセッサ・インターフェースの状態を次のようにモニタする。

- ー ピッコロは、レジスタ再充填されるのを、あるいは出力 F I F O が使えるエントリを持つのを、待つ態勢に入った。
- ー R O B 内のスペースが不十分であるか、出力 F I F O 内の事項(items)が不

十分であるかの理由で、コプロセッサ・インターフェースがビジー待ち状態(busy-waiting)にある。

これらの両方の条件が検出されると、ピッコロは、その状態レジスタに D ビットをセットし、中止し、A R M コプロセッサの命令を拒絶し、A R M は未定義命令トラップにはまる。

このデッドロック状態の検出により、少なくともプログラマにこのような条件が生じたことを知らせ、また失敗の正確な点（位置）を知らせることができるシステムが構成される。プログラマは、A R M とピッコロのプログラム・カウンタとレジスタを読めばよい。尚、強調しておくが、デッドロックが生じるのは、間違つたプログラムあるいはピッコロの状態を変造するシステム部分がある場合だけである。デッドロックは、データが少なすぎることや「オーバーロード」によつて生じることはない。

A R M からピッコロを制御するのに使用できるいくつかの操作があり、それらは C D P 命令によつて提供される。これらの C D P 命令は、A R M が優先状態にある時に受け付けられる。そうでないと、ピッコロは C D P 命令を拒絶し、A R M は未定義命令トラップにはまる。以下の操作が使用可能である。

- ー Reset (リセット)
- ー Enter State Access Mode (状態アクセスモードに入る)
- ー Enable (イネーブル)
- ー Disable (ディスエーブル)

ピッコロは、P R E S E T 命令によつてソフトウェア内でプリセットされる。

```
P R E S E T      ; Clear Piccolo's state
```

(ピッコロの状態をクリアする)

この命令は、次のようにエンコードされる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	1110	0000	0000	0000	PICCOLOI	000	0	0000
------	------	------	------	------	----------	-----	---	------

この命令が実行されると、次のことが生じる。

- すべてのレジスタが空き（再充填の態勢）の印になる。
- 入力R O Bがクリアされる。
- 出力F I F Oがクリアされる。
- ループ・カウンタがリセットされる。
- ピッコロは中止状態に入る（そしてS 2のHビットがセットされる）。

P R E S E T命令の実行には、いくつかのサイクル（この実施の形態では、2から3）が必要である。実行されている間に、以下のピッコロ上で実行されるべきA R Mコプロセッサ命令がビジー待ちになる。

状態アクセスモードにおいて、ピッコロの状態は、S T C及びL D C命令（以下のA R Mからのピッコロ状態アクセスについての説明を参照）を使って退避され復元される。状態アクセスモードに入るには、P S T A T E命令がまず実行されなければならない。

P S T A T E ;      Enter State Access Mode

(状態アクセスモードに入る)

この命令は次のようにエンコードされる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	1110	0001	0000	0000	PICCOLOI	000	0	0000
------	------	------	------	------	----------	-----	---	------

実行されると、P S T A T E命令は、

- ピッコロを中止し（すでに中止されているのでなければ）、Eビットをピッ

コロの状態レジスタにセットする。

ー ピッコロを状態アクセスモードに構成する。

P S T A T E 命令の実行が終わるまでにはいくつかのサイクルがある。というのは、ピッコロの命令パイプラインは中止する前に汲み出されなければならないからである。実行中、ピッコロ上で実行される次の A R M コプロセッサ命令がビジー待ちになる。

P E N A B L E 及び P D I S A B L E 命令は、高速コンテキスト切替えに使用される。ピッコロがディスエーブルされると、専用レジスタ 0 と 1 だけが ( I D

レジスタ、状態レジスタ) アクセス可能となり、それも優先モードからだけである。これ以外の状態へアクセスすると、またユーザモードからアクセスすると、A R M 未定義命令例外が生じる。ピッコロをディスエーブルすると、実行が中止される。ピッコロは、実行を中止すると、状態レジスタに E ビットをセットして応答する。

ピッコロをイネーブルするには、P E N A B L E 命令を実行する。

P E N A B L E ;      Enable Piccolo

この命令は次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND		1110		0010		0000		0000		PICCOLOI		000		0		0000															

ピッコロをディスエーブルするには、P D I S A B L E 命令を実行する。

P D I S A B L E ;      Disable Piccolo

この命令は次のようにエンコードされる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
COND		1110		0011		0000		0000		PICCOLOI		000		0		0000															

この命令が実行されると、次のことが生じる。



- ー ピッコロの命令パイプラインが空になる(drain)。
- ー ピッコロは中止して、状態レジスタにHビットをセットする。
- ー このセクションは、ピッコロのデータパスを制御するピッコロ命令セット（集合）に言及する。各命令は32ビットの長さである。これらの命令は、ピッコロ命令キャッシュから読み出される。

命令セットのデコードは、非常に直線的である。最初6ビット（26から31）が主要オペコードを与え、22から25までが、いくつかの特定の命令のためのマイナーなオペコードを提供する。灰色の影となっているコードは、現在未使用のものであり、拡張用として使える（それらは現時点で指示された値を含ん

でいなければならない）。

11の主要命令クラスがある。これは、いくつかのサブクラスのデコードを簡単にするため、命令にファイルされた主要オペコードに完全に対応するものではない。



3 3 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 9 8 7 6 5 4 3 2 1 0  
 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0

0	OPC	F	S	DEST	S	R	SRC1	SRC2
		D			I	I		
1	000	OPC	F	S	S	R	SRC1	SRC2
		D			I	I		
1	001	0	O	F	S	S	R	SRC1
		P		D		I	I	SRC2
1	0011							
1	010	OPC	F	S	S	R	SRC1	SRC2_SHIFT
		D			I	I		
1	011	00	F	S	S	R	SRC1	SRC2_SEL
		D			I	I		COND
1	011	01						
1	011	1	O	F	S	S	R	SRC1
		P		D	I	I		SRC2_SEL
								COND
1	10	0	O	S	F	S	S	R
		P	a	D	I	I	SRC1	SRC2_MULA
1	10	1		D				
1	10	1	O	1	F	S	S	R
		P		D	I	I	SRC1	SRC2_REG
							0	A
							0	R
							2	SCALE
1	110							
1	11100	F	S	DEST			IMMEDIATE_15	
		D						
1	11101							
1	11110	0	RFIELD_4		0	R	SRC1	#INSTRUCTIONS_8
					I			
1	11110	1	RFIELD_4				#LOOPS_13	#INSTRUCTION_8
1	11111	0	OPC				REGISTER_LIST_16	SCALE
1	11111	100					IMMEDIATE_16	COND
1	11111	101					PARAMETERS_21	
1	11111	11	O					
		P						

上の表にある命令には、以下の名前がついている。

Standard Data Operation (標準データ操作)

Logical Operation (論理操作)

Conditional Add/Subtract (条件付加算／減算)

Undefined (未定義)

Shifts (シフト)

Select (選択)

Undefined (未定義)

Parallel Select (並列選択)

Multiply Accumulate (乗算累算)

Undefined (未定義)

Multiply Double (乗算ダブル)

Undefined (未定義)

Move Signed Immediate (符号付即値移動)

Undefined (未定義)

Repeat (反覆)

Repeat (反覆)

Register List Operation (レジスタ・シフト操作)

Branch (ブランチ)

Renaming Parameter Move (リネーム・パラメータ移動)

Halt/Break (中止／中断)

命令の各クラスのフォーマットは、次のセクションに詳しく述べてある。ソース及び目的オペランド領域は、ほとんどの命令において共通であり、レジスタ・リマッピングと同様、別のセクションに述べてある。

ほとんどの命令は2つのソースオペランドSource1, Source2を必要とする。

Source1 (SRC1) オペランドは、次の7ビット・フォーマットを持つ。

18	17	16	15	14	13	12
Size	Refill	Register Number				Hi/Lo

この領域の要素は、次の意味を持つ。

- － Size－読み出すオペランドのサイズを示す（1＝32ビット、0＝16ビット）。
- － Refill－レジスタが読み出された後、空きの印になり、R.O.Bから再充填できることを示す。
- － Register Number－32ビット、16ビットレジスタのどちらのレジスタを読み出すべきかかエンコードする。
- － Hi/Low－16ビット読み出しに対して、32ビットレジスタのどちらの半分を読み出すべきかを示す。32ビットオペランドに対してセットされた場合は、レジスタの2つの16ビット半分が入れ換えられなければならないことを示す。

Size	Hi/Lo	Portion of Register Accessed
0	0	Low 16 bits
0	1	High 16 bits
1	0	Full 32 bits
1	1	Full 32 bits, halves swapped

レジスタのサイズは、レジスタ番号に接尾辞を付けることによってアセンブラによって特定される。下位16ビットなら、. l、上位16ビットなら、. h、32ビットの上下の16ビットを入れ換えるなら、. x。

一般のソース2（SCR2）は、次の3つの12ビット・フォーマットの1つを持つ。

11	10	9	8	7	6	5	4	3	2	1	0
0	S2	R2	'Register Number				Hi/Lo	SCALE			
1	0	ROT		IMMED_8							
1	1	IMMED_6						SCALE			

図 4 は、選択されたレジスタの適切な半分をピッコロのデータパスにスイッチするための Hi / Lo ビット及び Size ビットに応答するマルチプレクサ構成を示す。Size ビットが 16 ビットであれば、符号拡張回路がデータパスの高次ビットに適切な 0 または 1 を入れる。

最初のエンコーディングは、ソースをレジスタとして指定し、その領域は、SC R1 指定子 (specifier) と同じエンコーディングを持つ。SCALE 領域は、ALU の結果に適用されるべきスケールを指定する。

SCALE				Action
3	2	1	0	
0	0	0	0	ASR #0
0	0	0	1	ASR #1
0	0	1	0	ASR #2
0	0	1	1	ASR #3
0	1	0	0	ASR #4
0	1	0	1	RESERVED
0	1	1	0	ASR #6
0	1	1	1	ASL #1
1	0	0	0	ASR #8
1	0	0	1	ASR #16
1	0	1	0	ASR #10
1	0	1	1	RESERVED
1	1	0	0	ASR #12
1	1	0	1	ASR #13
1	1	1	0	ASR #14
1	1	1	1	ASR #15

8ビット即値(immediate)は、回転(rotate)エンコーディングによって、3  
2ビット即値を生成し、それが、8ビット値及び2ビット回転(rotate)によっ  
て表現される。次の表は、8ビット値XYから生成される即値を示す。

ROT	IMMEDIATE
00	0x000000XY
01	0x0000XY00
10	0x00XY0000
11	0xXY000000

6 ビット即値エンコーディングにより、6 ビット符号無し即値（範囲 0 から 63）を、ALU の出力に提供されるスケールと共に使用することができる。

一般のソース 2 エンコーディングは、ほとんどの命令変種に共通である。この規則には例外が少しあり、それがソース 2 エンコーディングの限定されたサブセットをサポートするか、あるいは、それを少し変形させる。

- Select Instructions. （選択命令）
- Shift Instructions. （シフト命令）
- Parallel Operations. （並列操作）
- Multiply Accumulate Instructions. （乗算累算命令）
- Multiply Double Instructions. （乗算ダブル命令）

選択命令は、レジスタ又は 16 ビット符号無し即値であるオペランドをサポートするだけである。スケールは無効である。それは、これらのビットは命令の条件領域によって使用されるからである。

	11	10	9	8	7	6	5	4	3	2	1	0
SRC2_SEL	0	S2	R2	Register number				Hi/Lo	COND			
	1	1	IMMED_6						COND			

シフト命令は、16 ビットレジスタ又は 5 ビット符号無し即値である 1 から 31 のオペランドをサポートするだけである。結果のスケールは無効である。



	11	10	9	8	7	6	5	4	3	2	1	0
SRC2_SHIFT	0	0	R2	Register number				Hi/Lo	0	0	0	0
	1	0	0	0	0	0	0	IMMED_5				

並列操作の場合、レジスタがオペランドのソースとして指定されていれば、32ビット読み出しが行われなければならない。即値エンコーディングは、並列操作については、少し違った意味を持つ。これにより、即値は、32ビットオペランドの16ビット半分の両方に複製できる。並列操作には少し制限のある範囲のスケールが使用できる。

	11	10	9	8	7	6	5	4	3	2	1	0
SRC2_PARALLEL	0	1	R2	Register number				Hi/L o	SCALE_PAR			
	1	0	ROT	IMMED_8								
	1	1	IMMED_6						SCALE_PAR			

6ビット即値が使用された場合、常に、32ビット量の半分の両方に複製される。8ビット即値が使用された場合は、それが複製されるのは、それが32ビット量の上半分に回転されるべきであると回転 (rotate) が示している時だけである。

ROT	IMMEDIATE
00	0x000000XY
01	0x0000XY00
10	0x00XY00XY
11	0xXY00XY00

並列選択操作にはスケールは無効である。スケール領域は、これらの命令では

0 にセットされる。

乗算累算命令では、8ビット回転即値を指定することはできない。領域のビット10は、どのアキュムレータを使用すべきかを指定する部分となる。ソース2は、16ビットオペランドとして意味される。

	11	10	9	8	7	6	5	4	3	2	1	0
SRC2_MULA	0	A0	R2	Register number				Hi/ Lo		SCALE		
	1	A0	IMMED_6							SCALE		

乗算ダブル命令は、定数を使用することができない。16ビットレジスタだけが指定できる。この領域のビット10は、どのアキュムレータを使用すべきかを指定する部分となる。

	11	10	9	8	7	6	5	4	3	2	1	0
SRC2_MULD	0	A0	R2	Register number				Hi/ Lo	SCALE			

命令のうちいくつかは、常に32ビット操作（例えば、ADDADD）を含み、その場合、サイズ・ビットは、1にセットされ、Hi/Loビットは、場合によっては32ビットオペランドの2つの16ビット半分を交換するのに使用することができる。また、いくつかの命令は、常に16ビット操作（例えば、MUL）を含み、サイズビットは0に設定されなければならない。Hi/Loビットは、レジスタのどちらの半分が使用されるかを選択する（見えないサイズビットは明らかなものと仮定する）。乗算・累算命令は、ソース・アキュムレータと目的レジスタを独立に指定することができる。これらの命令においては、Sizeビットは、ソースアキュムレータを指定するのに使用され、サイズビットは、命令タイプによって0と暗示される。

16ビット値が（A又はBバスを介して）使用される場合、それは、自動的に

32ビット量に符号拡張される。48ビットレジスタが（A又はBバスを介して

）読みだされる場合、下の32ビットだけがバスに現れる。それは、どの場合でも、ソース1、ソース2は、32ビット値に変換されるからである。バスCを使用する累算命令だけがアキュムレータレジスタの48ビット全部にアクセスすることができる。

再充填ビットがセットされていれば、レジスタは使用后、空きの印になり、普通の再充填メカニズムによってR0Bから再充填される（R0Bについてのセクションを参照）。ピッコロは、再充填が行われる以前にソースオペランドとしてレジスタが再び使用されないかぎり、止まらない。再充填されたデータが有効になる前のサイクルの最小数（最善の場合で、データはR0Bの先頭で待っている）は、1か2である。従つて、再充填されたデータは、再充填要求の次の命令には使わない方がよい。もし、次の2つの命令上でオペランドの使用を避けることができるのであれば、その方がよい。というのは、これにより、より深いパイプライン導入上での性能損失を防ぐことになるから。

再充填ビットは、レジスタ番号に接尾辞“<sup>^</sup>”を付けることによってアセンブラで指定される。空きの印のついたレジスタのセクションは、レジスタのオペランドに依存する。各レジスタの2つの半分は、独立に、再充填の印をつけることもできる（例えば、X0.1<sup>^</sup>は、X0の下半分だけを再充填することになり、X0<sup>^</sup>は、X0全体を再充填することになる）。48ビットレジスタの上「半分」（ビット47：16）が再充填されると、16ビットのデータがビット31：16に書き込まれ、ビット47まで符号拡張される。

同じレジスタを2回再充填しようと（例えば、ADD X1, X0<sup>^</sup>, X0<sup>^</sup>）しても、再充填は1度しか行われぬ。アセンブラは、ADD X1, X0, X0<sup>^</sup>という文法しか許可すべきではない。

レジスタ読み出しが、レジスタの再充填以前に試みられると、ピッコロは、レジスタが再充填されるまでまち状態で止まる。レジスタが再充填の印になると、レジスタは再充填の値が読まれる以前に更新され、その結果、UNPREDICTABLE（予想不可）となる（例えば、ADD X0, X0<sup>^</sup>, X1は予想不可。なぜなら、X0については再充填の印であるから、X0と

X 1 の合計で埋めることになる)。

4 ビットスケール領域は 1 4 のスケールタイプをエンコードする。

- A S R # 0 , 1 , 2 , 3 , 4 , 6 , 8 , 1 0
- A S R # 1 2 から 1 6
- L S L # 1

並列 M a x / M i n 命令は、スケールを提供しないので、ソース 2 の 6 ビット定数変種は使用されない (アセンブラにより 0 にセットされる)。

R E P E A T 命令内で、レジスタのリマッピングがサポートされ、R E P E A T が、ループを解かないままレジスタの移動「窓」にアクセスすることができる。これについて、以下、詳しく説明する。

目的オペランドは、次の 7 ビットフォーマットを持つ。

25	24	23	22	21	20	19
F	SD	HL	DEST			

この基本エンコーディングには 1 0 の変種がある。

## アセンブラ ニューモニック

25 24 23 22 21 20 19

Dx	1	0	1	0	Dx		
Dx^	2	1	1	0	Dx		
Dx.l	3	0	0	0	Dx		
Dx.l^	4	1	0	0	Dx		
Dx.h	5	0	0	1	Dx _		
Dx.h^	6	1	0	1	Dx		
未定義		0	1	1	0000		
.l (レジスタ書き戻しなし, 16ビット)	7	1	1	1	0	0	00
"" (レジスタ書き戻しなし, 32ビット)	8	1	1	1	0	1	00
.l^ (16-ビット) 出力	9	1	1	1	1	0	00
^ (32-ビット) 出力	10	1	1	1	1	1	00

レジスタ番号 (D x) は 16 のレジスタのどれがアドレスされているかを示す。Hi / Lo ビットと Size ビットは、一緒になって、各 32 ビットレジスタへ 16 ビットレジスタのペアとしてアドレスする。Size ビットは、どのようにしたら適切なフラグが、命令タイプに定義されているように、セットされるかを定義するもので、結果がレジスタバンク及び／又は出力 FIFO に書き込まれるか否かには関係しない。これにより、比較及び同様の命令の構成ができる。命令の累算クラスのある加算は、結果をレジスタに書き戻さなければならない。

エンコード	レジスタ書込	FIFO 書込	V フラグ
1	レジスタ全体に書く	書込なし	32-ビット オーバーフロー
2	レジスタ全体に書く	32ビット書込	32-ビット オーバーフロー
3	ロー16ビットを Dx.lに書く	書込なし	16-ビット オーバーフロー
4	ロー16ビットを Dx.lに書く	ロー16ビットに 書込	16-ビット オーバーフロー
5	ロー16ビットを Dx.hに書く	書込なし	16-ビット オーバーフロー
6	ロー16ビットを Dx.hに書く	ロー16ビットに 書込	16-ビット オーバーフロー
7	書込なし	書込なし	16-ビット オーバーフロー
8	書込なし	書込なし	32-ビット オーバーフロー
9	書込なし	ロー16ビットに 書込	16-ビット オーバーフロー
10	書込なし	32ビット書込	32-ビット オーバーフロー

どの場合でも、レジスタへの書き戻し又は出力FIFOへの挿入以前の操作の結果は、48ビット量である。2つの場合がある。

書き込みが16ビットならば、48ビット量は、下の16ビット[15:0]を選択することによって、16ビットに減る。命令が飽和すれば、値は、 $-2^{15}$ から $2^{15}-1$ の範囲に飽和される。次に16ビット値が指示されたレジスタに書き戻され、また、書き込みFIFOビットがセットされれば、出力FIFOに書き戻される。出力FIFOに書き込まれた場合、それは、次の16ビット値が書き込まれるまで保持される。次の16ビットが書き込まれると、それらの値はペアとなって、単一32ビット値として出力FIFOに入れられる。

32ビットの書き込みならば、48ビットは、下の32ビット[31:0]を選んで32ビット量に減る。

32ビット書き込みでも、48ビット書き込みでも、命令が飽和すれば、48ビット値は $-2^{31}-1$ から $2^{31}$ の範囲の32ビット値に変換される。飽和すると、

- アキュムレーへの書き戻しが行われると、48ビット全部が書き込まれる。
- 32ビットレジスタへの書き戻しが行われると、ビット[31:0]が書き込まれる。
- 出力FIFOへの書き戻しが行われる場合も、やはりビット[31:0]が書き込まれる。

目的サイズは、レジスタ番号の後の、lまたは、hによって、アセンブラ内で指定される。レジスタへの書き戻しが全く行われない場合は、レジスタ番号は意味がなくなるので、目的レジスタを省略して、レジスタへの書き込み無しとするか、あるいは、^を使って、出力FIFOだけへの書き込みを指示する。例えば、SUB, X0, Y0は、CMP X0, Y0と等価であり、ADD ^, X0, Y0は、X0 + Y0の値を出力FIFOに入れる。

出力FIFOに値を入れる空がない場合は、ピッコロは、空ができるまで待機する。

16ビット値、例えば、ADD X0, h ^, X1, X2が書き出されると、その値は、第2の16ビット値が書かれるまでラッチされる。次にこれら2つの値は結合されて、32ビット数として出力FIFOに入れられる。最初に書き込まれる16ビット値は、常に32ビットワードの下半分に現れる。出力FIFOに入れられたデータは、16又は32ビットデータとしての印がつき、endianessをbig endianシステム上で訂正することができる。

32ビット値が2つの16ビット書き込みの間に書き込まれると、その動作は未定義になる。

REPEAT命令内で、レジスタ・リマッピングがサポートされ、REPEATは、ループを解く(unroll) ことなしにレジスタの移動「窓」にアクセスす

ることができる。以下、これについて詳しく説明する。

本発明の好ましい実施の形態において、REPEAT命令は、レジスタ・オペランドがループ内で特定される方法を変更するメカニズムを提供する。このメカニズムの下で、アクセスするレジスタは命令内のレジスタ・オペランドとレジスタバンクのオフセットの機能によって決定される。オフセットは、プログラム可能な方法で変更でき、各命令ループの最後で変更されるのが好ましい。このメカニズムは、X, Y, Zバンク内にあるレジスタ上で独立に動作することができる。好ましい実施の形態では、この機能はAバンク内のレジスタには使用できない。

論理レジスタ、物理レジスタという概念を使用することができる。命令オペランドは論理レジスタを参照し、これらは、特定のピッコロレジスタ10を同定する物理レジスタ・レファレンスにマップされる。すべての操作は、再充填も含み、物理レジスタ上で動作する。レジスタ・リマッピングが生じるのは、ピッコロ命令ストリームサイドだけであり、ピッコロにロードされるデータは常に物理レジスタを目的とし、リマッピングは行われない。

リマッピングのメカニズムについて、以下、図5を参照して説明する。図5は、ピッコロ・コプロセッサ4の多数の内部構成要素を示すブロック図である。メモリからARMコア2によって検索されるデータ項目は、リオーダ・バッファ12に入れられ、ピッコロレジスタ10は、先に図2を参照した方法で、リオーダ・バッファ12から再充填される。キャッシュ6に記憶されているピッコロの命令は、ピッコロ4内の命令デコーダ50に渡されることによって、ピッコロ・プロセッサ・コア54に渡される前にデコードされる。ピッコロ・プロセッサ・コア54は、先に図3を参照して述べた乗算器／加算器回路20と、累算／退出回路22と、スケール／飽和(saturate)回路24とを備える。

命令デコーダ50がREPEAT命令によって同定された命令ループの一部を構成する命令を扱っていて、且つ、そのREPEAT命令が多数のレジスタのリマッピングを行うことが必要であると指示した場合は、レジスタ・リマッピング論理52が使用されて、必要なリマッピングが行われる。レジスタ・リマッピング論理52は、命令デコーダ50の一部であると考えて良い。ただし、当業者に



は明らかなように、レジスタ・リマッピング論理52は、命令デコーダ50に対して全く別のものとして提供されてもかまわない。

典型的な命令は、その命令にとって必要なデータ項目を含むレジスタを同定する1つまたは2つ以上のオペランドを備える。例えば、典型的な命令は、2つのソースオペランドと1つの目的ペランドを含むことができ、その命令が必要とするデータ項目を含む2つのレジスタと、その命令の結果を入れるべきレジスタを同定する。レジスタ・リマッピング論理52は、命令デコーダ50から命令のオペランドを受け取るが、それらは論理レジスタ・レファレンスを同定する。論理レジスタ・レファレンスに基づき、レジスタ・リマッピング論理は、物理レジスタのリマッピングをすべきかどうかを決定し、必要なら、物理レジスタ・レファレンスにリマッピングを適用する。また、リマッピングを適用すべきではないと決定された場合は、論理レジスタ・レファレンスが物理レジスタ・レファレンスとして提供される。リマッピングを行う好ましい方法については、後で、詳しく説明する。

レジスタ・リマッピング論理からの各出力物理レジスタ・レファレンスは、ピッコロ・プロセッサ・コア54に渡されることによって、プロセッサ・コアが、物理レジスタ・レファレンスによって同定される特定のレジスタ10内のデータ項目に命令を適用できるようにする。

好ましい実施の形態によるリマッピングのメカニズムによれば、レジスタの各バンクは、2つのセクション、つまりその中でレジスタがリマップされるセクションと、レジスタがリマッピング無しで元のレジスタ・レファレンスを保持するセクションとの2つのセクションに割ることができる。好ましい実施の形態において、リマップされたセクションは、リマップされているレジスタ・バンクの下から開始される。

このリマッピングのメカニズムは多数のパラメータを使用し、これらのパラメータについては、図6を参照しながら、詳細に説明する。図6は、様々なパラメータがレジスタ・リマッピング論理52によっていかに使用されるかを示すブロック図である。尚、これらのパラメータは、リマップされているバンク内の点、

例えば、バンクの下からの相対的値を与えられている。

レジスタ・リマッピング論理52は、2つの主要論理ブロック、つまりRemap

(リマップ) ブロック56とBase Update (ベース更新) ブロック58とからなると考えることができる。レジスタ・リマッピング論理52は、論理レジスタ・レファレンスに加えられるべきオフセット値を提供するベース・ポインタを使用する。このベース・ポインタの値は、ベース更新ブロック58によってリマップ・ブロックに提供される。

B A S E S T A R T 信号を使用して、ベースポインタの初期値を定義することができる。例えば、典型的には、ゼロであるが、他の値を指定することもできる。このB A S E S T A R T 信号は、ベース更新ブロック58内のマルチプレクサ60に渡される。命令ループの最初の繰り返しで、B A S E S T A R T 信号は、マルチプレクサ60によって、記憶エレメント66に渡され、ループのその後の繰り返しでは、次のベース・ポインタ値がマルチプレクサ60によって記憶エレメント66に渡される。

記憶エレメント66の出力は、現在のベース・ポインタ値としてリマップ論理56に渡され、またベース更新論理58内の加算器62の入力の1つにも渡される。加算器62は、ベース・インクリメント値を提供するB A S E I N C 信号を受け取る。加算器62は、記憶エレメント66によって供給される現在のベース・ポインタ値を、B A S E I N C 値分だけインクリメントし、結果をモジュロ回路64へ渡すようになっている。

また、モジュロ回路は、B A S E W R A P 値を受け取り、この値を加算器62からの出力ベース・ポインタ信号と比較する。インクリメントされたベース・ポインタ値がB A S E W R A P 値以上であれば、その新しいベース・ポインタがラップラウンドされて、新しいオフセット値となる。モジュロ回路64の出力は、記憶エレメント66に記憶されるべき次のベース・ポインタ値となる。この出力はマルチプレクサ60に提供され、そこから、記憶エレメント66に提供される。

しかしながら、この次のベース・ポインタ値は、R E P E A T 命令を管理する

ループ・ハードウェアからBASEUPDATE信号を記憶エレメント66が受け取らないうちは、記憶エレメント66に記憶できない。BASEUPDATE信号は、ループ・ハードウェアによって周期的に生成され、例えば、命令ループが反復されるごとに、生成される。BASEUPDATE信号を記憶エレメント

66が受け取ると、記憶エレメントは、以前のベース・ポインタ値にマルチプレクサ60から提供される次のベース・ポインタ値を上書きする。このように、リマップ論理58に供給されるベース・ポインタ値は、新しいベース・ポインタ値に変わる。

レジスタバンクのリマップされたセクション内でアクセスされるべき物理レジスタは、命令のオペランド内に含まれる論理レジスタ・レファレンスに、ベース更新論理58によって提供されるベース・ポインタ値を加えることによって決定される。この加算を行うのは加算器68であり、その出力は、モジュロ回路70に渡される。好ましい実施の形態において、モジュロ回路70は、レジスタ・ラップ値を受け取り、加算器68からの出力信号（論理レジスタ・レファレンスとベース・ポインタ値の和）がレジスタ・ラップ値を越えると、その結果へ、リマップされた領域の下でラップ（wrap）が行われる。モジュロ回路70の出力は、マルチプレクサ72に提供される。

REGCOUNT値がリマップ・ブロック56内の論理74に提供され、リマップされるべきバンク内のレジスタの個数を同定する。論理74は、このREGCOUNT値を論理レジスタ・レファレンスと比較し、比較の結果により、制御信号をマルチプレクサ72に渡す。マルチプレクサ72は、その2つの入力で、論理レジスタ・レファレンスとモジュロ回路70からの出力（リマップされたレジスタ・レファレンス）を受け取る。本発明の好ましい実施の形態において、論理レジスタ・レファレンスがREGCOUNT値より小さければ、論理74は、マルチプレクサ72にリマップされたレジスタ・レファレンスを物理レジスタ・レファレンスとして出力させる。ただし、もし、論理レジスタ・レファレンスがREGCOUNT値以上であれば、論理74は、マルチプレクサ72に論理レジスタ・レファレンスを直接、物理レジスタ・レファレンスとして出力させる。

先に述べたように、好ましい実施の形態において、リマッピング・メカニズムを引き起こすのはREPEAT命令である。後で、より詳しく述べるように、REPEAT命令は、ハードウェアで4つのゼロサイクルループを提供する。これらのハードウェア・ループは、図5に命令デコーダ50の部分として図示されている。命令デコーダ50がキャッシュ6から命令を要求する度に、キャッシュは

その命令を命令デコーダに戻し、それにより、命令デコーダは、戻された命令がREPEAT命令であるかどうか判断する。もしそれであれば、ハードウェア・ループの1つが、そのREPEAT命令を扱うように構成される。

各繰り返し命令は、ループ内の命令の数と、ループを繰り返す回数（定数またはピッコロ・レジスタから読み出される）を指定する。2つのオペコードREPEATとNEXTがハードウェアループの定義用に提供され、NEXTオペコードは単に区切りとして使用されるだけで、命令としてアセンブルはされない。REPEATがループの頭に行き、NEXTがループの最後を区切ることによって、アセンブラはループ・本体内の命令の数を数えることができる。好ましい実施の形態において、REPEAT命令は、レジスタ・リマッピング論理52が使用するREGCOUNT, BASEINC, BASEWRAP, REGWRAPパラメータのようなリマッピング・パラメータを含むことができる。

レジスタ・リマッピング論理によって使用されるリマッピング・パラメータを記憶する多数のレジスタを提供することができる。これらのレジスタ内で、前もって定義されたリマッピング・パラメータの多数のセット（集合）を提供することができる一方、いくつかのレジスタはユーザ定義リマッピング・パラメータを記憶するために残される。REPEAT命令と共に指定されたリマッピング・パラメータが、前もって定義されたリマッピング・パラメータの1つと等しい場合、適当なREPEATエンコーディングが使用され、これにより、マルチプレクサ等が適当なリマッピング・パラメータをレジスタから直接にレジスタ・リマッピング論理へ提供する。一方、リマッピング・パラメータが前もって定義されたリマッピング・パラメータのどれとも等しくない場合は、アセンブラがRemapping Parameter Move (RM OV) 命令を生成する。これにより、ユーザ定義レジス

タ・リマッピング・パラメータの構成が可能となり、RMOV命令の後にREPEAT命令が続く。好ましくは、ユーザ定義リマッピング・パラメータは、RMOV命令によって、そのようなユーザ定義リマッピング・パラメータを記憶すべく残されていたレジスタに入れられ、マルチプレクサは、それらのレジスタの内容をレジスタ・リマッピング論理に渡すようプログラムされる。

好ましい実施の形態において、REGCOUNT, BASEIN, BASEWRAP, REGWRAPパラメータは、以下のチャートに示された値の1つを取る。

パラメータ	説明
REGCOUNT	これは、0、2、4、又は8で、リマッピングを行う16ビットレジスタの数を同定する。REGCOUNT未満のレジスタはリマップされ、REGCOUNT以上のレジスタは直接アクセスされる。
BASEINC	これは、16ビットレジスタをいくつ使用して、各ループ繰り返しの最後でベース・ポインタがインクリメントされるかを定義する。好ましい実施の形態では、1、2、又は4の値を取れるが、実際には所望すれば、負の値を含む他の値も取れる。
BASEWRAP	これはベース計算のシーリングを決める。ベース・ラッピング・モジュラスは、2、4、8の値を取れる。
REGWRAP	これは、リマップ計算のシーリングを決める。レジスタ・ラッピング・モジュラスは2、4、8の値を取れる。REGWRAPは、REGCOUNTと等しく選択することができる。

図6に戻り、リマップ・ブロック56によって様々なパラメータが使用される例を次に示す（この例では、論理及び物理レジスタ値は、特定バンクに対する相対値である。）

```

if (Logical Register (論理レジスタ) < REGCOUNT)

    Physical Register(物理レジスタ) = (Logical Register(論理レジスタ) + Base (ベース)) MOD REGCOUNT

else

    Physical Register(物理レジスタ) = Logical Register (論理レジスタ)

```

end if

ループの最後で、ループの次の繰り返しが始まる前に、次のベース・ポインタ更新がベース更新論理58によって行われる。

$$\text{Base} = (\text{Base} + \text{BASE INC}) \text{ MOD } \text{BASE WRAP}$$

リマッピング・ループの最後でレジスタ・リマッピングが打ち切られ、すべてのレジスタは物理レジスタとしてアクセスされる。好ましい実施の形態において、

1つのリマッピングREPEATだけがどの時点においてもアクティブである。ループは、ネストされたままであるが、ある特定の時点で1つだけがリマッピング変数を更新してよい。ただし、所望するなら、リマッピング繰り返しはネストできるようにする。

本発明の好ましい実施の形態に基づくリマッピング・メカニズムを使用した結果としてのコード密度に関して達成される効果を示すために、以下、典型的なブロック・フィルタ・アルゴリズムについて説明する。まず、ブロック・フィルタ・アルゴリズムの原則について、図7を参照しながら説明する。図7に示されているように、アキュムレータ・レジスタA0は、多数の乗算操作の結果を累算するように備えられている。この乗算操作というのは、係数c0とデータ項目d0との乗算、係数c1とデータ項目d1との乗算、係数c2とデータ項目d2との乗算などである。レジスタA1は、乗算操作の同様のセットの結果を累算していくが、今度は、係数がずれて、c0とd1、c1とd2、c2とd3と組み合わせの乗算になる。同様に、レジスタA2は、係数値を更にずらして、c0とd2、c1とd3、c2とd4といった組み合わせの乗算の結果を累算する。このシフト、乗算、累算のプロセスが、繰り返され、その結果がレジスタA3に入れられる。

本発明の好ましい実施の形態に基づくレジスタ・リマッピングを使用しないと、ブロック・フィルタ命令を実行するには、次のような命令ループが必要となる。

; start with 4 new data values

(4つの新しいデータ値で始める)

Z E R O {A 0 - A 3} ; Zero the accumulators

(アキュムレータをゼロにする)

R E P E A T Z 1 ; Z 1 = (number of coeffs/4)

(係数の個数)

; do the next four coefficients, on the first time around:

(第1回目に、次の4つの係数で行う)

; a0 += d0\*c0+ d1\*c1+ d2\*c2+ d3\*c3

; a1 += d1\*c0+ d2\*c1+ d3\*c2+ d4\*c3

; a2 += d2\*c0+ d3\*c1+ d4\*c2+ d5\*c3

; a3 += d3\*c0+ d4\*c1+ d5\*c2+ d6\*c3

MUL A A0, X0.l<sup>^</sup>, Y0.l, A0 ;a0 += d0\*c0, and load d4

(そして d 4 をロードする)

MUL A A1, X0.h , Y0.l, A1 ;a1 += d1\*c0

MUL A A2, X1.l , Y0.l, A2 ;a2 += d2\*c0

MUL A A3, X1.h , Y0.l<sup>^</sup>, A3 ;a3 += d3\*c0, and load c4

(そして c 4 をロードする)

MUL A A0, X0.h<sup>^</sup>, Y0.h, A0 ;a0 += d1\*c1, and load d5

(そして d 5 をロードする)

MUL A A1, X1.l , Y0.h, A1 ;a1 += d2\*c1

MUL A A2, X1.h , Y0.h, A2 ;a2 += d3\*c1

MUL A A3, X0.l , Y0.h<sup>^</sup>, A3 ;a3 += d4\*c1, and load c5

(そして c 5 をロードする)

MUL A A0, X1.l<sup>^</sup>, Y1.l, A0 ;a0 += d2\*c2, and load d6

(そして d 6 をロードする)

MUL A A1, X1.h , Y1.l, A1 ;a1 += d3\*c2

MUL A A2, X0.l , Y1.l, A2 ;a2 += d4\*c2

MUL A A3, X0.h , Y1.l<sup>^</sup>, A3 ;a3 += d5\*c2, and load c6

(そして c 6 をロードする)

MUL A A0, X1.h<sup>^</sup>, Y1.h, A0 ;a0 += d3\*c3, and load d7

(そして d 7 をロードする)

MUL A A1, X0.l , Y1.h, A1 ;a1 += d4\*c3

MUL A A2, X0.h , Y1.h, A2 ;a2 += d5\*c3

MUL A A3, X1.l , Y1.h<sup>^</sup>, A3 ;a3 += d6\*c3, and load c7

(そして c 7 をロードする)

NEXT

この例において、データ値はレジスタの X バンクに入れられ、係数値はレジスタの Y バンクに入れられる。第 1 ステップとして、4 つのアキュムレータ・レ



ジスタ A 0 , A 1 , A 2 , A 3 はゼロにセットされる。アキュムレータ・レジスタがリセットされると、命令ループが開始され、このループは R E P E A T 命令及び N E X T 命令によって区切られる。Z 1 の値は、この命令ループが繰り返される回数を示し、また後で述べる理由により、この回数は、実際には、係数の個数 ( c 0 , c 1 , c 2 など) を 4 で割った数に等しい。

命令ループには 1 6 の乗算累算命令 ( M U L A : multiply accumulate instructions) があり、1 回目の繰り返しが終わると、その結果、レジスタ A 0 , A 1 , A 2 , A 3 は、R E P E A T 命令と第 1 の M U L A 命令との間で上のコードで示される計算の結果を含む。乗算累算操作がどのように動作するかを示すために、最初の 4 つの M U L A 命令を考えることにする。最初の命令によって、X バンク・レジスタ・ゼロの最初のすなわち下の 1 6 ビット内のデータ値と、Y バンク・レジスタ・ゼロ内の下の 1 6 ビットとが掛け合わされ、その結果がレジスタ A 0 に加えられる。これと同時に、X バンク・レジスタ・ゼロの下の 1 6 ビットが再充填の印になり、レジスタのこの部分に新しいデータ値が再充填できることを示す。このように印がつき、図 7 から明らかなように、データ項目 d 0 が係数 c 0 で乗算されると (これは最初の M U L A によって表される)、d 0 は、ブロック・フィルタ命令の残り部分では不要になり、新しいデータ値で置き換えられる。

次に、第 2 の M U L A によって、X バンク・レジスタ・ゼロの第 2 のすなわち上の 1 6 ビットと、Y バンク・レジスタ・ゼロの下の 1 6 ビットとが掛け合わされ (これは、図 7 における、 $d_1 \times c_0$  を表す)。同様に、第 3、第 4 の M U L A 命令が、 $d_2 \times c_0$ 、及び  $d_3 \times c_0$  の乗算を行う。図 7 から明らかなように、これらの 4 つの計算が行われると、係数 C 0 は不要となり、レジスタ Y 0 . 1 は、再充填の印がつき、他の係数 ( c 4 ) で上書きできるようになる。

次の 4 つの M U L A 命令は、それぞれ、 $d_1 \times c_1$  ,  $d_2 \times c_1$  ,  $d_3 \times c_1$  ,  $d_4 \times c_1$  の計算を表す。 $d_1 \times c_1$  の計算が終了すると、d 1 は不要になるので、レジスタ X 0 . h は再充填ビットの印がつく。同様に、4 つの計算すべてが終了すると、係数 c 1 は不要になるので、レジスタ Y 0 . h は再充填用の印が

つ

く。同様に、次の4つのMUL A命令は、 $d2 \times c2$ 、 $d3 \times c2$ 、 $d4 \times c2$ 、 $d5 \times c2$ の計算に対応し、最後の4つの計算は、 $d3 \times c3$ 、 $d4 \times c3$ 、 $d5 \times c3$ 、 $d6 \times c3$ の計算に対応する。

上記の実施の形態において、レジスタはリマップできず、各乗算操作は、オペランドで指定される特定レジスタによって明示的に再生されなければならない。16のMUL A命令の実行が終了すると、係数 $c4$ から $c7$ 及びデータ項目 $d4$ から $d10$ まで、命令ループを繰り返すことができる。また、ループは、繰り返し1回につき4つの係数値で操作するので、係数値の個数は、4の倍数でなければならない、 $Z1 = \text{係数} / 4$ 個の計算が行われる。

本発明の好ましい実施の形態におけるリマッピング・メカニズムを使用することによって、命令ループは飛躍的に減らすことができ、4つの乗算累算命令を含むだけになる。さもないと16の乗算累算命令が必要になる。このリマッピング・メカニズムを使用すると、コードは以下のように書くことができる。

; start with 4 new data values

(4つの新しいデータ値で始める)

Z E R O {A0-A3} ; Zero the accumulators

(アキュムレータをゼロにする)

R E P E A T Z1, X++ n4 w4 r4, Y++ n4 w4 r4; Z1=(number of coefficients)

(係数の個数)

; Remapping is applied to the X and Y banks.

(リマッピングがXとYバンクに適用される。)

; Four 16 bit registers in these banks are remapped.

(これらのバンク内の4つの16ビット・レジスタがリマップされる。)

; The base pointer for both banks is incremented by one on each

; iteration of the loop.

(両方のバンクのベース・ポインタが、ループの繰り返しごとに1つインクリメントされる。)

; The base pointer wraps when it reaches the fourth register in the bank.

(ベース・ポインタは、バンク内の第4のレジスタに到達するとラップ(wrap)する。)

M U L A A0, X0.l<sup>^</sup>, Y0.l, A0 ;a0 += d0\*c0, and load d4

(そしてd4をロードする)

M U L A A1, X0.h, Y0.l, A1 ;a1 += d1\*c0

M U L A A2, X1.l, Y0.l, A2 ;a2 += d2\*c0

M U L A A3, X1.h, Y0.l<sup>^</sup>, A3 ;a3 += d3\*c0, and load c4

(そしてc4をロードする)

N E X T ;go round loop and advance remapping

(ループを1周し、リマップを進める)

先に述べたのと同様に、第1のステップで、4つのアキュムレータ・レジスタA0-A3をゼロにセットする。次に、R E P E A TオペコードとN E X Tオペコードによって区切られる命令ループに入る。R E P E A T命令は、以下のよう多数のパラメータを持つ。

X++ : レジスタのXバンクに、B A S E I N Cが'1'であることを示す

。 n 4 : REGCOUNT が ' 4 ' であり、従って、最初の 4 つの X バンクレジスタ X 0 . 1 から X 1 . h がリマップされることを示す。

w 4 : レジスタの X バンクに、BASEWRAP が ' 4 ' であることを示す。

。 Y + + : レジスタの Y バンクに、BASEINC が ' 1 ' であることを示す。

。 n 4 : REGCOUNT が ' 4 ' であり、従って、最初の 4 つの Y バンクレジスタ Y 0 . 1 から Y 1 . h がリマップされることを示す。

w 4 : レジスタの Y バンクに、BASEWRAP が ' 4 ' であることを示す。

。 r 4 : レジスタの Y バンクに、REGWRAP が ' 4 ' であることを示す。

尚、Z 1 の値は、先行技術の例では、係数の個数 / 4 に等しくなるが、ここでは、係数の個数と等しくなる。

命令ループの最初の繰り返しで、ベースポインタの値はゼロであり、リマッピングはない。ただし、次にループが実行される時は、X バンクも Y バンクもベース・ポインタの値は ' 1 ' であるから、オペランドは次のようにマップされる。

X 0 . 1 は X 0 . h になる

X 0 . h は X 1 . 1 になる

X 1 . 1 は X 1 . h になる

X 1 . h は X 0 . 1 になる ( BASEWRAP が ' 4 ' だから )

Y 0 . 1 は Y 0 . h になる

Y 0 . h は Y 1 . 1 になる

Y 1 . 1 は Y 1 . h になる

Y 1 . h は Y 0 . 1 になる ( BASEWRAP が ' 4 ' だから )

従って、2 回目の繰り返しでは、本発明のリマッピングを含まない先に述べた例における第 5 から第 8 番目の M U L A 命令によって示される計算を、4 つの M U L A 命令が実際に行うことがわかる。同様に、3 回目、4 回目のループの繰り返し

返しでは、先行技術コードの第9から第12番目、そして第13から第16番目のMULA命令によって実行された計算が行われる。

従って、上記コードは、先行技術のコードと全く同様のブロック・フィルタ・アルゴリズムを行うわけだが、ループ本体内のコード密度を4倍に改善している。つまり、先行技術では16の命令が必要であったのに比較して、4つの命令で済む。

本発明の好ましい実施の形態に基づくレジスタ・リマッピング技術を使用することによって、以下のような利点が得られる。

1. コード密度を改善する。
2. 場合によっては、レジスタを空きとして印をしてからピッコロのリオーダ・バッファによって再充填されるまでのレイテンシー(latency)を隠すこともできる。これは増えるコードサイズを捨ててアンローリングループによって実現される。
3. アクセスされるべきレジスタの数を変化させることができる。ループ繰り返し実行数を変化させることによって、アクセスされるレジスタの数を変化させることができる。
4. アルゴリズム開発を簡単にすることができる。適当なアルゴリズムについて、プログラマはアルゴリズムのn番目の段に対する1つのコードを生成して、レジスタ・リマッピングを使用して、その公式をデータのスライディング・セットに適用することができる。

上記レジスタ・リマッピング・メカニズムは、本発明の範囲から離れることなく、ある程度の変形が可能であることが明らかになるであろう。例えば、レジスタ10のバンクは、プログラマによって命令オペランドに指定される以上の物理レジスタを提供することができる。これらの余分のレジスタは直接的にはアクセスできないが、レジスタ・リマッピング・メカニズムでは、これらのレジスタを使用することができる。例えば、先に出した例を考えてみよう。レジスタのXバンクに、プログラマの使える32ビットレジスタが4つあり、従って8つの16ビットレジスタが論理レジスタ・レファレンスによって指定することができる。レ

ジスタのXバンクが、実際には、例えば6つの32ビットレジスタから成る場合、プログラマにとって直接アクセスできない16ビットレジスタが余分に4つあることになる。しかしながら、これらの4つのレジスタは、リマッピング・メカニズムによって使用可能となり、データ項目の記憶のための付加的レジスタを提供する。

以下のアセンブラ・シンタクス（文法）を使用することができる。

>>は、論理右シフト、又は、シフト・オペランドが負であれば、左シフトを意味する（下の<1scale>を参照）。

->>は、算術右シフト、又は、シフト・オペランドが負であれば、左シフトを意味する（下の<scale>を参照）。

RORは、右回転を意味する。

SAT(a)は、aの飽和値を意味する（目的レジスタのサイズによって、16ビット又は32ビットで飽和する）。特に、16ビットで飽和するために、+0x7ffffより大きいどんな値も+0x7ffffで置き換えられ、-0x800より小さいどんな値も-0x8000で置き換えられる。32ビット飽和は、同様に、極限值+0x7fffffffと-0x80000000がある。目的レジスタが48ビットである場合も、飽和は32ビットで行われる。

ソース・オペランド1は、次のフォーマットの1つを取ることができる。

<src1>は、[Rn;Rn.1;Rn.h;Rn.x][^]の短縮形として使用される。別の言い方をするなら、ソース・スペシファイアの7ビットはすべて有効であり、レジスタは32ビット値として（希望すれば、交換される）、また

は符号拡張した16ビット値として読まれる。アキュムレータに取っては、下の32ビットだけが読まれる。“^”は、レジスタ再充填を指定する。

<src\_\_16>は、[Rn.1;Rn.h][^]の短縮形として使用される

16ビット値だけが読まれる。

<src\_\_32>は、[Rn;Rn.x][^]の短縮形として使用される。3

2 ビット値だけが読まれ、上半分及び下半分は希望すれば交換できる。

ソース・オペランド2は、次のフォーマットの1つを取ることができる。

<src2>は、3つのオプションの短縮形として使用される。

—  $[Rn!Rn.1!Rn.h!Rn.x] [^]$  の形のソース・レジスタ、  
 プラス最終結果のスケール (<scale>)。

— オプションでシフトされた8ビット定数 (<immed\_\_8>)、ただし、  
 最終結果のスケールはない。

— 6ビット定数 (<immed\_\_6>)、プラス、最終結果のスケール (<scale>)。

<src2 \_\_maxmin>は、<src2>と同じであるが、ただし、スケールは許可されない。

<src2 \_\_shift>シフト命令は、<src2>の限定的サブセットを提供する詳細は上記を参照。

<src2 \_\_par>      <src2 \_\_shift>用である。

第3のオペランドを指定する命令に対して：

<acc>は、4つのアキュムレータ・レジスタ  $[A0!A1!A2!A3]$  の  
 いずれかを示す短縮形。48ビットすべてが読まれる。再充填は指定されない。

目的レジスタは次のフォーマットを持つ：

<dest>      これは、 $[Rn!Rn.1!Rn.h!.1!]$   $[^]$  の短縮形。

“.” の拡張はない。

レジスタ全部が書かれる (アキュムレータの場合は、48ビット)。レジスタへの書き戻しが必要ない場合は、使用されるレジスタは重要でない。アセンブラが、目的レジスタの省略をサポートし、書き戻しの必要がないこと、又は“.”つまり、書き戻しは必要ないが結果が16ビット量であるかのようにフ

ラグをセットすべきであることを示す。^は、値が出力FIFOに書き込まれることを示す。

<scale>      これは、代数スケールの数を表す。14のスケールが使用できる。

ASR # 0, 1, 2, 3, 4, 6, 8, 10

ASR # 12 から 16

LSL # 1

<imm\_8> これは、符号無し 8 ビット即値を表す。これは、0、8、16、又は 24 シフトで左回転された 1 バイトから成る。従って、0xYZ0000000、0x00YZ00000、0x0000YZ000、0x0000000YZの値が、任意の YZ に対してエンコードできる。回転は、2 ビット量としてエンコードされる。

<imm\_6> これは、符号無し 6 ビット即値を表す。

<PARAMS> これは、レジスタ・リマッピングを指定し、次のフォーマットを持つ： <BANK><BASIC>n<RENUMBER>w<BASEWRAP>

<BANK>	[X;Y;Z] を取れる
<BASEINC>	[++;+1;+2;+4] を取れる
<RENUMBER>	[0;2;4;8] を取れる
<BASEWRAP>	[2;4;8] を取れる

<cond>という表現は、以下の条件コードの任意の 1 つの短縮形である。尚、エンコーディングは、ARM と少し異なる。それは、符号無し LS 及び HI コードは、より役立つ符号付きオーバーフロー／アンダーフローのテストで置き換えられているからである。V フラグ及び N フラグは、ピッコロ上で、ARM とは違う方法でセットされるので、条件テストからフラグ・チェックへの翻訳も、ARM とは異なる。

0000	EQ	Z = 0	最後の結果はゼロであった。
0001	NE	Z = 1	最後の結果は非ゼロであった。
0010	CS	C = 1	shift/MAX 操作の後で使用される。
0011	CC	C = 0	



0100	MI/LT	N=1	最終結果が負であった。
0101	PL/GE	N=0	最終結果が正であった。
0110	VS	V=1	最終結果で符号付きオーバーフロー/飽和
0111	VC	V=0	最終結果でオーバーフロー/飽和なし
1000	VP	V=1 & N=0	最終結果でオーバーフロー正
1001	VN	V=1 & N=1	最終結果でオーバーフロー負
1010	未使用		
1011	未使用		
1100	GT	N=0 & Z=0	
1101	LE	N=1 ; Z=1	
1110	AL		
1111	未使用		

ピッコロが扱うのは符号付き量であるから、符号無しLS及びHI条件は、落とされ、オーバーフローの方向を記述するVPとVNで置き換えられている。ALUの結果は48ビット幅であるから、MIとLTが、同様にPLとGEが同じ機能を行う。

すべての操作は、特に注意書のない限り、符号付きである。

第1条件コード及び第2条件コードは、それぞれ、次のものから成る。

N 負  
Z ゼロ  
C キャリー/符号無しオーバーフロー  
V 符号付きオーバーフロー

算術命令は、並列命令と「フル幅」命令の2つに分けることができる。「フル幅」命令というのは、一次フラグをセットするだけであるのに対して、並列オペレータは、結果の上16ビット半分と下16ビット半分とに基づき、一次フラグと2次フラグをセットする。

N, Z, Vフラグは、スケールを適用した後に、目的に書き込まれる前に、ALUの結果に基づいて計算される。ASRは常に、結果を記憶するのに必要なビット数を減らす、ASLだと、それを増やす。これを避けるために、ピッコロ

は、ASLスケールが適用された場合、48ビットの結果を削って、ゼロ検出及びオーバーフローが行われるビット数を制限する。

Nフラグの計算は、符号付き算術計算が行われると推定して、行われる。それは、オーバーフローが起きた場合、結果の最上位ビットはCフラグかNフラグであり、それは、入力オペランドが符号付きか符号無しかによるからである。

Vフラグは、選択された目的に結果を書き込んだ結果、精度の損失があるか否かを示す。書き戻しが選択されなかった場合も、「サイズ」は含まれており、オーバーフロー・フラグは正しくセットされる。オーバーフローが起きるのは、次の場合である。

－ 結果が、 $-2^{15}$  から  $2^{15}-1$  の範囲にないのに16ビットレジスタに書き込んだ場合。

－ 結果が、 $-2^{31}$  から  $2^{31}-1$  の範囲にないのに32ビットレジスタに書き込んだ場合。

並列加算／減算命令は、結果の上半分及び下半分に独立にN，Z，Vフラグをセットする。

アキュミュレータに書き込みを行うと、32ビットレジスタに書き込まれたかのように、Vフラグがセットされる。

飽和絶対命令 (SABS) も、入力オペランドの絶対値が指定された目的に合わない、オーバーフロー・フラグをセットする。

キャリー・フラグは、加算と減算命令によりセットされ、MAX/MIN，SABS、CLB命令によって「バイナリー」フラグとして使用される。乗算操作を含む他のすべての命令は、(単数または複数の) キャリー・フラグを保存する。

加算と減算操作については、キャリーは、ビット31又はビット15又は目的が32ビット幅であるか16ビット幅であるかの、結果によって生成される。

標準的算術命令は、フラグのセット方法によって、多くのタイプに分類することができる。

加算命令、減算命令の場合、Nビットがセットされると、すべてのフラグが保存される。Nビットがセットされないと、フラグは、次のように更新される。

Z がセットされるのは、フル 48 ビット結果が 0 だった場合。

N がセットされるのは、フル 48 ビット結果にビット 47 のセットがあった場合（負だった場合）。

V がセットされるのは：

目的レジスタが 16 ビットであり、符号付き結果が 16 ビットレジスタに合わない ( $-2^{15} \leq x < 2^{15}$  の範囲にない) 場合

目的レジスタが 32 / 40 ビットレジスタであり、符号付き結果が 32 ビットに合わない場合

<dest> が 32 又は 40 ビットレジスタである場合で C フラグがセットされるのは、<scr1> と <scr2> を合計してビット 31 からキャリーがある時、又は、<scr1> から <scr2> を減算してビット 31 から借り (borrow) が生じない時 (ARM 上と同じキャリー)。<dest> が 16 ビットレジスタである場合で C フラグがセットされるのは、合計のビット 15 からキャリーがある時。

2 次フラグ (SZ, SN, SV, SC) は保存される。

48 ビットレジスタから乗算又は累算を行う命令の場合。

Z がセットされるのは、フル 48 ビット結果が 0 だった場合。

N がセットされるのは、フル 48 ビット結果にビット 47 のセットがあった場合（負だった場合）。

V がセットされるのは：(1) 目的レジスタが 16 ビットであり、符号付き結果が 16 ビットレジスタに合わない ( $-2^{15} \leq x < 2^{15}$  の範囲にない) 場合、(2) 目的レジスタが 32 / 48 ビットレジスタであり、符号付き結果が 32 ビットに合わない場合

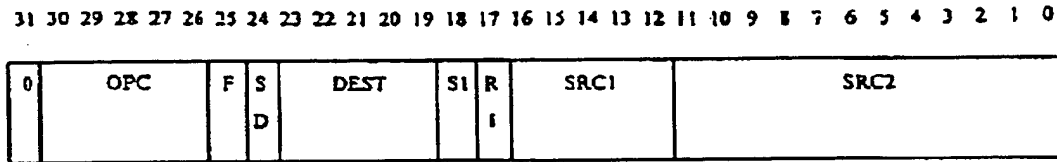
C は保存される。

2 次フラグ (SZ, SN, SV, SC) は保存される。

論理操作、並列加算及び減算、max 及び min、シフトなどを含むその他の命令は、以下のようにカバーされる。

加算命令、減算命令は、2 つのレジスタを加算又は減算し、結果をスケールして、レジスタに戻して記憶させる。オペランドは、符号付き値として扱われる。

不飽和変種に対するフラグ更新は、オプションであり、Nを命令の最後に付け足すことによって抑制することもできる。



OPCは、命令のタイプを指定する。

```

100N0    dest=(src1 + src2) (->> scale) (,N)
110N0    dest=(src1 - src2) (->> scale) (,N)
10001    dest=SAT((src1 + src2) (->> scale))
11001    dest=SAT((src1 - src2) (->> scale))
01110    dest=(src2 - src1) (->> scale)
01111    dest=SAT((src2 - src1) (->> scale))
101N0    dest=(src1 + src2 + Carry) (->> scale) (,N)
111N0    dest=(src1 - src2 + Carry-1) (->> scale) (,N)

```

ニューモニックス：

```

100N0    ADD {N}    <dest>, <src1>, <src2> {,<scale>}
110N0    SUB {N}    <dest>, <src1>, <src2> {,<scale>}
10001    SADD       <dest>, <src1>, <src2> {,<scale>}
11001    SSUB      <dest>, <src1>, <src2> {,<scale>}
01110    RSB       <dest>, <src1>, <src2> {,<scale>}
01111    SRSB      <dest>, <src1>, <src2> {,<scale>}
101N0    ADC {N}    <dest>, <src1>, <src2> {,<scale>}
111N0    SBC {N}    <dest>, <src1>, <src2> {,<scale>}

```

アセンブラは以下のオペコードをサポートする

```

CMP    <src1>, <src2>
CMN    <src1>, <src2>

```

CM Pは、レジスタ書き込みディスエーブル (disabled) のフラグをセットす

る減算であり、CMNは、レジスタ書き込みディスエーブルのフラグをセットする加算である。

フラグ：

これについては、上記の通り。

含める理由

ADCは、shift/MAX/MIN操作に続いてレジスタの下にキャリーを挿入するのに使える。また、32/32割算を行うのにも使用される。さらに、拡張精密加算を提供する。Nビットを加算することによって、フラグを細かく制御することができ、特にキャリーを制御できる。これにより、1ビットにつき2サイクルで、32/32ビット割算ができる。

飽和加算及び減算が、G. 729などに必要である。

カウンタのインクリメント/デクリメント。RSBは計算シフト ( $x = 32 - x$  が普通の操作) に使える。飽和RSBは、飽和否定 (G. 729で使用される) に必要である。

加算/減算累算命令は、累算及びスケール/飽和を伴う加算及び減算を行う。乗算累算命令と違って、アキュムレータ番号は、目的レジスタと独立に指定することはできない。目的レジスタの下2ビットは、累算に使う48ビットアキュムレータの番号、accを与える。従って、ADDA X0, X1, X2, A0及びADDA A3, X1, X2, A3は有効であるが、ADDA X1, X1, X2, A0は無効である。このクラスの命令では、結果はレジスタに書き戻されなければならない、目的領域の書き戻し無しエンコーディングは許可されない。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	0	0	1	0	Sa	F	S	DEST	SI	R	SRC1	SRC2
		P					D			1		
		C										

OPCは、命令のタイプを指定する。以下において、accは (DEST [1 : 0]) である。Saビットは、飽和を示す。

動作 ( O P C ) :

```

0    dest = {SAT} (acc + (src1 + src2)) {->> scale}
1    dest = {SAT} (acc + (src1 - src2)) {->> scale}

```

ニューモニック :

```

0    {S} ADDA <dest>, <src1>, <src2>, <acc> {,<scale>}
1    {S} SUBA <dest>, <src1>, <src2>, <acc> {,<scale>}

```

コマンドの前の S は飽和を示す。

フラグ :

上記を参照

含める理由

A D D A (加算累算) 命令は、1 サイクルにつき、整数アレーの 2 ワードとアキュムレータ (例えば、それらの平均を見つけるのに) の和を取るのに使える。S U B A (減算累算) 命令は、差の和を計算するのに (例えば相関のために) 使え、2 つの別個の値を減算して、その差を第 3 のレジスタに加える。

<acc> とは異なる <dest> を使用することによって、丸め (rounding) をともなう加算をすることもできる。例えば、 $X_0 = (X_1 + X_2 + 16384) >> 15$  は、16384 を A0 に保持しながら 1 サイクルで行うことができる。丸め付定数加算は、A D D A X0, X1, #16384, A0 で行うことができる。

ビットの正確な導入には :

$\text{sum of } ((a\_i * b\_i) >> k)$  (一般的には TrueSpeech で使用される)

標準ピッコロ・コードは以下のようなになる :

```

M U L    t1,    a__0, b__0, ASR#k
A D D    ans, ans, t1
M U L    t2, a__1, b__1, ASR#k
A D D    ans, ans, t2

```

このコードには 2 つの問題がある。1 つは長すぎることで、もう 1 つは、加算が 48 ビット精密加算ではなくガードビットが使用できないこと。これに対処するには、A D D A を使うことである。

```

M U L    t1, a__0, b__0, ASR#k
M U L    t2, a__1, b__1, ASR#k
A D D A  ans, t1, t2, ans

```

これにより、25%のスピードアップが得られる、48ビット精度が保持される。

並列命令における加算／減算は、32ビットレジスタに対（ペア）で保持される2つの符号付き16ビット量で行われる。一次条件コードフラグは、最上位16ビットの結果からセットされ、二次フラグは、下位半分から更新される。これらの命令のソースとして指定できるのは32ビットレジスタだけであるが、値は、ハーフワード交換できる。各レジスタの個々の半分は、符号付き値として扱われる。計算及びスケールリングは、精度損失無しで行われる。従って、A D D A D D X 0, X 1, X 2, A S R # 1 は、X 0 の上半分及び下半分における正しい平均を生成する。各命令にはオプションナル飽和が提供され、それには、S a ビットをセットする。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	0	OPC	Sa	F	S	DEST	SI	R	SRC1	SRC2
					D			1		

O P C が操作を定義する。

動作（O P C）：

```

000  dest.h=(src1.h + src2.h)->> {scale} ,
      dest.l=(src1.l + src2.l)->> {scale}
001  dest.h=(src1.h + src2.h)->> {scale} ,
      dest.l=(src1.l - src2.l)->> {scale}
100  dest.h=(src1.h - src2.h)->> {scale} ,
      dest.l=(src1.l + src2.l)->> {scale}
101  dest.h=(src1.h - src2.h)->> {scale} ,
      dest.l=(src1.l - src2.l)->> {scale}

```

S a ビットがセットされている場合、各和／差分は独立に飽和する。

ニューモニック：

000 {S} ADDADD <dest>, <src1\_32>, <src2\_32> {,<scale>}

001 {S} ADDSUB <dest>, <src1\_32>, <src2\_32> {,<scale>}

100 {S} SUBADD <dest>, <src1\_32>, <src2\_32> {,<scale>}

101 {S} SUBSUB <dest>, <src1\_32>, <src2\_32> {,<scale>}

コマンドの前の S は飽和を示す。

アセンブラは以下のものもサポートする

CMN CMN <dest>, <src1\_32>, <src2\_32> {,<scale>}

CMN CMP <dest>, <src1\_32>, <src2\_32> {,<scale>}

CMPC MN <dest>, <src1\_32>, <src2\_32> {,<scale>}

CMPC CMP <dest>, <src1\_32>, <src2\_32> {,<scale>}

書き戻しなしの標準命令によって生成される。

フラグ

C がセットされるのは、2 つの上の 16 ビット半分を加算する時に、ビット 15 のキャリーがある場合。

Z がセットされるのは、上の 16 ビット半分の和が 0 である場合。

N がセットされるのは、上の 16 ビット半分の和が負である場合。

V がセットされるのは、上の 16 ビット半分の符号付き 17 ビット和が 16 ビットに当てはまらない（ポスト・スケール）場合。

S Z, S N, S V, S C が、同様に、下の 16 ビット半分に対してセットされる。

含める理由

並列加算及び減算命令は、単一 32 ビットレジスタに保持される複素数を操作するのに使用でき、FFT カーネルで使用される。また、16 ビットデータのベクトルの単純な加算／減算にも使え、1 サイクルで 2 つの要素を処理することができる。

ブランチ（条件付き）命令は、制御フローにおける条件付き変更を行うことを



可能とする。ピッコロは、取られたブランチを実行するのに3サイクル使う。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	11111	100	000	IMMEDIATE_16	COND
---	-------	-----	-----	--------------	------

#### 動作

一次フラグに基づき<cond>が保持されれば、オフセットによるブランチ。

オフセットは、符号付き16ビット番号のワードである。この時、オフセットの範囲は、-32768から+32767ワードに制限される。

アドレス計算は次のようにされる。

目的アドレス = ブランチ命令アドレス + 4 + オフセット

ニューモニック：

B <cond> <destination\_label>

フラグ：

影響されない

含める理由：

殆どのルーチンで非常に役立つ。

条件付き加算又は減算命令は、条件付きでsrc2をsrc1へ加算または減算する

。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	0010	O P C	F S D	DEST	S I R	SRC1	SRC2
---	------	-------------	-------------	------	-------------	------	------

O P C が命令のタイプを指定する。

動作 ( O P C ) :

```

0   if(carry set)temp=src1-src2 else temp=src1+src2
    dest = temp  {->> scale}

1   if(carry set)temp=src1-src2 else temp=src1+src2
    dest = temp  {->> scale} BUT if scale is a shift left
                                (もしスケールが左シフトなら)

    then the new value of carry (from src1-src2 or src1+src2) is
    shifted into the bottom.

    (src1-src2 又は src1+src2からの) キャリーの新しい値がボトムにシ
    フトされる)

```

ニューモニック：

```

0   CAS      <dest>, <src1>, <src2> {,<scale>}
1   CASC     <dest>, <src1>, <src2> {,<scale>}

```

フラグ：

上記参照

含める理由

条件付き加算または減算命令により、効率のよい除算コードを構成することができる。

例 1： X0にある32ビット符号無し値を、X1にある16ビット符号無し値で割る (X0 < (X1 << 16) 且つ X1.h = 0 と仮定する)。

```

LSL  X1, X1, #15      ; 除数をシフトアップする
SUB  X1, X1, #0        ; キャリーフラグをセットする
REPEAT #16
CASC X0, X0, X1, LSL#1
NEXT

```

ループの最後で、X0.1は除算の商を保持する。余りは、キャリーの値に従って、X0.hから復元される。

例 2： X0にある32ビット正の値を、X1にある32ビット正の値で割り、早く終了する。

```

MOV      X2, #0          ; 商をクリアする
LOG      Z0, X0          ; ビット数X0分シフトできる
LOG      Z1, X1          ; ビット数X1分シフトできる
SUBS     Z0, Z1, Z0      ; X1がシフトして1がマッチする
BLT      div__end        ; X1 > X0 なので答は0
LSL      X1, X1, Z0      ; マッチした場合
ADD      Z0, Z0, #1      ; 行うべきテストの回数
SUBS     Z0, Z0, #0      ; キャリーをセットする
REPEAT   Z0
CAS      X0, X0, X1, LSL#1
ADCN     X2, X2, X2

```

## NEXT

div\_\_end

最後に、X2が商を保持し、余りは、X0から復元される。

カウント・リーディング・ビット命令により、データが正規化される。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

011011	F	S	DEST	SI	R	SRC1	101110000000
	D			I			

## 動作

destは、src1にある値が左にシフトされるべき場所数にセットされて、ビット31がビット30と異なるようにする。これは0-30の範囲の値であるが、例外として、src1が-1又は0の場合は、31が戻される。

## ニューモニック

CLB <dest>, <src1>

## フラグ

Zがセットされるのは、結果が0の時。

Nはクリアされる。

Cがセットされるのは、src1が-1又は0の時。

V は未使用。

含む理由：

正規化に必要なステップ

ピッコロの実行を止めるには、Halt及びBreakpoint命令がある。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	1111	11	OP	000000000000000000000000
---	------	----	----	--------------------------

O P C は命令のタイプを指定する。

動作 ( O P C )

0 ピッコロの実行が止められ、Haltビットがピッコロ状態レジスタにセットされる。

1 ピッコロの実行が止められ、Breakビットがピッコロ状態レジスタにセットされ、ARMが中断され、ブレークポイントに到達したことを知らせる。

ニューモニック

0 H A L T

1 B R E A K

フラグ

影響されない。

論理演算命令は、32又は16ビットレジスタ上で論理演算を行う。オペランドは、符号無し値として扱われる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	000	OPC	F	S	DEST	SI	R	SRC1	SRC2
			D				1		

O P C は、実行すべき論理操作をエンコードする。

動作 ( O P C ) :

```

00  dest = (src1 & src2) { - > > scale }
01  dest = (src1 | src2) { - > > scale }
10  dest = (src1 & ~src2) { - > > scale }
11  dest = (src1 ^ src2) { - > > scale }

```

ニューモニック :

```

00  AND  <dest>, <src1>, <src2>, {,<scale>}
01  ORR  <dest>, <src1>, <src2>, {,<scale>}
10  BIC  <dest>, <src1>, <src2>, {,<scale>}
11  EOR  <dest>, <src1>, <src2>, {,<scale>}

```

アセンブラが以下のオペコードをサポートする

```

TST  <src1>, <src2>
TEQ  <src1>, <src2>

```

T S T は、レジスタ書き込みがディスエーブルされた A N D である。T E Q はレジスタ書き込みがディスエーブルされた E O R である。

フラグ

Z がセットされるのは、結果が全て 0 の時。

N , C , V は保存される。

S Z , S N , S C , S V は保存される。

含む理由 :

スピーチ圧縮アルゴリズムは、情報をエンコードするために、パックされたビット領域を使用する。ビットマスク命令は、これらの領域の抽出／パック化を助ける。

M a x 及び M i n 操作命令は、最大及び最小操作を実行する。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	101	O P C	I	F	S D	DEST	S I	R I	SRC1	SRC2
---	-----	-------------	---	---	--------	------	--------	--------	------	------

O P C は命令のタイプを指定する。

動作 ( O P C ) :

0 dest = (src1 ≤ src2) ? src1 : src2

1 dest = (src1 > src2) ? src1 : src2

ニューモニック :

0 MIN <dest>, <src1>, <src2>

1 MAX <dest>, <src1>, <src2>

フラグ

Z がセットされるのは、結果が 0 の時。

N がセットされるのは、結果が負の時。

C Maxでは、src2 ≥ src1 (dest = src1 の場合) の時にセットされる。Minでは、src2 ≥ src1 (dest = src2 の場合) の時にセットされる。

V 保存される

含む理由 :

信号の強さを見るために、多数のアルゴリズムがサンプルをスキャンして、サンプルの絶対値の最大／最小を決める。これに、MAX, MIN操作が使用できる。信号の最初の最大値か最後の最大値のどちらを見つきたいかによって、オペランドsrc1及びsrc2は、交換することができる。

MAX X0, X0, #0は、X0を正の数に変換し下をクリップする。

MIN X0, X0, #255は、X0の上をクリップする。これは、グラフィック処理に役立つ。

並列命令におけるMAX, MIN操作は、並列16ビットデータ上で最大値、最小値操作を行う。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

0	111	O	I	F	S	DEST	S	R	SRC1	SRC2_PARALLEL
		P	...	D	D		I	I		

O P C は、命令のタイプを指定する。

動作 ( O P C ) :

```

0   dest.l = (src1.l <= src2.l) ? src1.l : src2.l
    dest.h = (src1.h <= src2.h) ? src1.h : src2.h
1   dest.l = (src1.l > src2.l) ? src1.l : src2.l
    dest.h = (src1.h > src2.h) ? src1.h : src2.h

```

ニューモニック：

```

0   MINMIN    <dest>, <src1>, <src2>
1   MAXMAX    <dest>, <src1>, <src2>

```

フラグ

Z がセットされるのは、結果の上 16 ビットがゼロの場合。

N がセットされるのは、結果の上 16 ビットが負の場合。

C Max: src2.h >= src1.h (dest=src1 の場合) の時にセットされる。

Min: src2.h >= src1.h (dest=src2 の場合) の時にセットされる。

V 保存される

S Z, S N, S C, S V は、同様に、下 16 ビット半分用にセットされる。

含む理由：

32 ビット Max, Min について。

Move Long Immediate Operation 命令により、レジスタは、どの符号付き 16 ビットの符号拡張値をセットされることができる。これらの命令のうち 2 つは、32 ビットレジスタに任意の値にセットすることができる（連続する高位半分と低位半分にアクセスすることによって）。レジスタ間の移動については、選択操作を参照。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	11100	F	S	DEST	IMMEDIATE_15	+/	000
		D				-	

MOV <dest>, # <imm\_16>

アセンブラは、MOV 命令を使用して非インターロック NOP 操作を提供することができる。つまり、NOP は、MOV, #0 と等価である。

フラグ

フラグは影響されない。

含む理由：

レジスタ／カウンタをイニシアライズする。

乗算累算操作命令は、符号付き乗算を行い、累算または退出(deaccumulation)、スケーリング及び飽和を伴う。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	10	OPC	Sa	F	S	DEST	A	R	SRC1	SRC2_MULA
					D		1	1		

OPC 領域は命令のタイプを特定する。

動作 (OPC)：

00 dest = (acc +(src1\*src2)) {—>> scale}

01 dest = (acc -(src1\*src2)) {—>> scale}

各場合、S a ビットがセットされていれば、結果は目的に書き込まれる前に飽和される。

ニューモニック：

00 {S} MULA <dest>, <src1 \_\_16>, <src2 \_\_16>, <acc> {,<scale>}

01 {S} MULS <dest>, <src1 \_\_16>, <src2 \_\_16>, <acc> {,<scale>}

コマンドの手前の S は飽和を示す。

フラグ：

上記を参照。

含む理由：

1 サイクル保持されたMULAがFIRコードに必要である。MULSは、FFTバタフライで使用される。また、MULAは、丸め(rounding)付き乗算に役立つ。例えば、 $A0 = (X0 * X1 + 16384) >> 15$ は、16384を別のアキュムレータ(例えばA1)に保持することによって、1つのサイクル



で行うことができる。FFTカーネルには異なった<dest>及び<acc>が必要である。

Multiply Double Operation命令は、符号付き乗算を行い、結果をダブルにしてから累算又は退出、スケーリング、飽和を行う。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

I	10	I	O	I	F	S	DEST	A	R	SRC1	O	A	R	SRC2	SCALE
		P				D		I	I			0	2		
		C													

O P C は命令のタイプを指定する。

動作 ( O P C )

0 dest=SAT ((acc + SAT (2\*src1\*src2)) {->> scale})

1 dest=SAT ((acc - SAT (2\*src1\*src2)) {->> scale})

ニューモニック :

0 SMLDA <dest>, <src1\_\_16>, <src2\_\_16>, <acc> [, <scale>]

1 SMLDS <dest>, <src1\_\_16>, <src2\_\_16>, <acc> [, <scale>]

フラグ :

上記参照

含む理由 :

MLD命令は、G. 729など、分数 (fractional) 算術を使用するアルゴリズムにとって必要である。殆どのDSPは、累算又は書き戻しの前に乗数の出力において1ビット左にシフトさせることのできる分数モードを提供する。これを特定命令としてサポートすることにより、プログラマにはより大きなフレキシビリティが与えられる。Gシリーズの基本操作のいくつかと同等の名前を以下に示す。

L \_\_msu=> SMLDS

L \_\_mac=> SMLDA

これらは、1ビット左シフトする時に乗数の飽和を利用する。一連の分数の乗算・累算が必要な場合、精度のロスなしに、MULAを使うことができ、その和は、33.14フォーマットで保持される。必要なら、左シフト及び飽和を最後

に利用して、1. 15フォーマットに変換することができる。

乗算演算命令は、符号付き乗算、及びオプションなスケール／飽和を行う。ソース・レジスタ（16ビットのみ）は、符号付き数として扱われる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

00011	O P C	F S D	DEST	S R 1	SRC1	SRC2
-------	-------------	-------------	------	-------------	------	------

O P C は命令のタイプを指定する。

動作（O P C）：

0 dest = (src1 \* src2) { - > > scale }

1 dest = SAT (src1 \* src2) { - > > scale }

ニューモニック：

0 MUL <dest>, <src1\_16>, <src2> {,<scale>}

1 SMUL <dest>, <src1\_16>, <src2> {,<scale>}

フラグ

上記を参照。

含む理由。

符号付き且つ飽和乗算は、多くの処理で必要となる。

Register List操作は、複数のレジスタのセット（集合）に操作を行う時に使用される。Empty and Zero命令は、ルーチンを始める前に、あるいはルーチンとルーチンとの間で、レジスタの選択をリセットするのに使用する。Output命令を使って、レジスタのリストの内容を出力F I F Oに記憶することができる。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	11111	0	OPC	00	REGISTER_LIST_16	SCALE
---	-------	---	-----	----	------------------	-------

O P C は命令のタイプを指定する。

動作（O P C）：

- 000 (k=0; k<16; k++)用  
if bit k of the register list is set then register k is marked as being empty. (レジスタリストのビットkがセットされると、レジスタkは空きの印になる。)
- 001 (k=0; k<16; k++)用  
if bit k of the register list is set then register k is set to contain 0. (レジスタリストのビットkがセットされると、レジスタkが0を含むようセットされる。)
- 010 未定義
- 011 未定義
- 100 (k=0; k<16; k++)用  
if bit k of the register list is set then (レジスタリストのビットkがセットされると、) (register k->>scale) is written to the output FIFO. (が出力FIFOに書き込まれる。)
- 101 (k=0; k<16; k++)用  
if bit k of the register list is set then (レジスタリストのビットkがセットされると、) (register k->>scale) is written to the output FIFO and register k is marked as being empty. (が出力FIFOに書き込まれ、レジスタkが空きの印になる。)
- 110 (k=0; k<16; k++)用  
if bit k of the register list is set then (レジスタリストのビットkがセットされると、) SAT (register k->>scale) is written to the output FIFO. (が出力FIFOに書き込まれる。)
- 111 (k=0; k<16; k++)用  
if bit k of the register list is set then (レジスタリストのビットkがセットされると、) SAT (register k->>scale) is written to the output FIFO and register k is marked as being empty. (が出力FIFOに書き込まれ、レジスタkが空きの印になる。)

000	EMPTY	<register_list>	
001	ZERO	<register_list>	
010	Unused		
011	Unused		
100	OUTPUT	<register_list>	{,<scale>}
101	OUTPUT	<register_list> ^	{,<scale>}
110	SOUTPUT	<register_list>	{,<scale>}
111	SOUTPUT	<register_list> ^	{,<scale>}

## フラグ

影響されない

## 例

EMPTY {A0, A1, X0 - X3}

ZERO {Y0 - Y3}

OUTPUT {X0 - Y1} ^

また、アセンブラはシンタクス（文法）をサポートする。

OUTPUT Rn

その場合、MOV ^, Rn 命令を使ってレジスタを1つ出力することになる。

EMPTY 命令は、空であるすべてのレジスタが有効データを含む（すなわち、空きでない）まで、止まっている。

リマッピング REPEAT ループ内では、レジスタ・リスト操作は使用されるべきでない。

OUTPUT 命令が出力用に指定することができるレジスタは8つまでである。

含む理由：

1つのルーチンが終了した後、次のルーチンは、ARMからデータを受け取るようすべてのレジスタが空きであることを期待する。これを逐行するために、EMPTY 命令が必要となる。FIR そのたのフィルタを実行する前に、すべてのアキュムレータ及び部分的結果がゼロにされなければならない。これには、

ZERO命令が助けとなる。これらの命令は、一連の単一レジスタ移動を置き換えることによってコード密度を改善するよう設計されている。OUTPUT命令は、一連のMOV, Rn命令を置き換えることによってコード密度を改善するべく含まれる。

リマッピング・パラメータ・移動命令RMOVが提供されるので、ユーザ定義のレジスタ・リマッピング・パラメータの構成を取ることができる。

命令エンコーディングは以下の通り。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	1111	101	00	ZPARAMS	YPARAMS	XPARAMS
---	------	-----	----	---------	---------	---------

各PARAMS領域は次のエントリから成る：

6 5 4 3 2 1 0

BASEWRAP	BASEINC	0	RENUMBER
----------	---------	---	----------

これらのエントリの意味を以下に示す。

パラメータ	説 明
RENUMBER	リマッピングを行う16ビットレジスタの数であって、値0, 2, 4, 8を取ることができる。 RENUMBER未満のレジスタはリマップされ、それ以上のレジスタは直接アクセスされる。
BASEINC	各ループの最後でベースポインタがインクリメントされる量。
BASEWRAP	ベースラッピング・モジュラスが取れる値は2, 4, 8。

ニューモニック：

RMOV<PARAMS>, [<PARAMS>]

<PARAMS>領域は次のフォーマットを取る。

<PARAMS> ::= <BANK> <BASEINC> n <RENUMBER> w <BASEWRAP>

<BANK>::= [X!Y!Z]

<BASEINC>::= [++!+1!+2!+4]

<RENUMBER>::= [0!2!4!8]

<BASEWRAP>::= [2!4!8]

RMOV命令の使用がリマッピングのアクティブ中だと、その挙動は、UNPREDICATABLE（予想不可）である。

フラグ

影響されない。

Repeat命令は、4つのゼロ・サイクル・ループをハードウェアで提供する。REPEAT命令は、新しいハードウェア・ループを定義する。ピッコロは、最初のREPEAT命令にハードウェア・ループ0を使用し、最初のrepeat命令に埋め込まれた（nested）REPEAT命令にハードウェア・ループ1を使用し、以下同様である。REPEAT命令は、どのループが使用されているかを指定する必要はない。REPEAT命令は厳密に埋め込まなければならない。深さ4を越える埋め込みを試みると、挙動は、予想不可となる。

各REPEAT命令は、（REPEAT命令の直後の）ループ内の命令の数を指定し、そのループを何回巡るかの回数（定数またはピッコロレジスタから読み込まれる）を指定する。

ループ内の命令の数が小さい（1又は2）場合、ピッコロはそのループをセットアップするために余分のサイクルを使っても良い。

ループ・カウントがレジスタ指定であれば、32ビットアクセスという意味になる（S1=1）が、下の16ビットだけが意味を持ち、その数は符号無しであるとされる。ループ・カウントがゼロの場合、ループの動作は未定義である。ループ・カウントのコピーが取られ、レジスタはループに影響せずに直接再利用（又は、再充填さえ）できる。

REPEAT命令は、ループ内でレジスタ・オペランドが指定される方法を変えるメカニズムを提供する。詳細は上記の通り。

ループ数がレジスタ指定されたREPEATのエンコーディング：

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	11110	0	RFIELD_4	00	0	R	SRC1	0000	#INSTRUCTIONS_8
						1			

固定されたループ数の R E P E A T のエンコーディング :

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	11110	1	RFIELD_4	#LOOPS_13	#INSTRUCTIONS_8

R F I E L D オペランドは、ループ内でどの 16 リマッピングパラメータ構成を使用すべきかを指定する。

RFIELD	BASEINC	RENUMBER {X,Y,Z}	BASEWRAP	ニューモニック
0	-	000	-	リマッピングなし
1	-	---	-	ユーザ定義 パラメータ
2	1	880	0	X++ n8, Y++ n8
3	1	800	0	X++ n8
4	1	440	0	X++ n4, Y++ n4
5	1	400	0	X++ n4
6	1	220	0	X++ n2, Y++ n2
7	1	200	0	X++ n2
8	2	880	0	X+2 n8, Y+2 n8
9	2	800	0	X+2 n8
10	2	440	0	X+2 n4, Y+2 n4
11	2	400	0	X+2 n4
12	1	820	0	X++ n8, Y++ n2
13	1	444	0	X++ n4, Y++ n4, Z++ n4
14	-	---	-	予約済
15	-	---	-	予約済

アセンブラは、ハードウェア・ループを定義するために R E P E A T と N E X T という 2 つのオペコードを提供する。R E P E A T はループの始めに行き、N E X T はループの最後を区切ることによって、アセンブラはループ本体内にある

命令の数を数えることができる。R E P E A T にとって必要なことは、ループの



数を定数あるいはレジスタとして指定すればよいだけである。例えば：

```
REPEAT      X0
MULA        A0, Y0. l, Z0. l, A0
MULA        A0, Y0. h^, Z0. h^, A0
NEXT
```

これは、2つのMULA命令をX0回実行する。また、

```
REPEAT      #10
MULA        A0, X0^, Y0^, A0
NEXT
```

は、10回乗算累算を行う。

アセンブラは、次のシンタクス（文法）をサポートする。

```
REPEAT #iterations [, <PARAMS>]
```

REPEATのために使用するリマッピング・パラメータを指定する。必要なリマッピング・パラメータが前もって定義されたパラメータのセットの1つと等しい場合は、適当なREPEATエンコーディングが使用される。そうでなければ、アセンブラはRMOVを生成してREPEAT命令に続くユーザ定義パラメータをロードする。RMOV命令及びリマッピング・パラメータ・フォーマットの詳細については前記を参照。

ループの繰り返し(iteration)の回数が0であれば、REPEATの動作はUNPREDICTABLE（予想不可）である。

命令数領域が0にセットされると、REPEATの動作は、予想不可である。

ループに1つの命令しかなく、その命令がブランチであれば、予想不可能の挙動をする。

REPEATループの範囲からそのループの外へのブランチは、予想不可である。

飽和絶対命令は、ソース1の飽和絶対値(saturated absolute)を計算する。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	10011							F	S	DEST					S1	R	SRC1					100000000000									
								D								1															

動作：

$dest = SAT((src1 \geq 0) ? src1 : -src1)$  . 値は常に飽和する。特に、 $0x80000000$ の絶対値は $0x7fffffff$ であり、 $0x80000000$ ではない。

ニューモニック：

S A B S <dest> , <src1>

フラグ

Zがセットされるのは、結果が0の時。

Nは保存される。

Cがセットされるのは、 $src < 0$  ( $dest = \_src1$ の場合)。

Vがセットされるのは、飽和が生じた時。

含む理由：

多くのDSPアプリケーションで役立つ。

選択(select)操作(条件付き移動)は、条件付きでソース1またはソース2を目的レジスタに移動させる。選択は、常に、移動と等価である。並列加算/減算の後で使用するための並列操作もある。

尚、両方のソースオペランドは、導入理由のための命令によっても読み出すことができるので、一方が空きであれば、そのオペランドが絶対的に必要であるかどうかに関係なく、命令は止まる。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	011							OPC	F	S	DEST					S1	R	SRC1					SRC2_SEL									
									D																							

OPCは、命令のタイプを指定する。

動作(OPC)：

- 00 If<cond>holds for primary flags then dest=src1  
 (もし<cond>が一次フラグのためであれば)  
 else dest=src2.
- 01 If<cond>holds for the primary flags then dest.h=src1.h  
 (もし<cond>が一次フラグのためであれば)  
 else dest.h=src2.h,  
 If<cond>holds for the secondary flags then dest.l=src1.l  
 (もし<cond>が二次フラグのためであれば)  
 else dest.l=src2.l.
- 10 If<cond>holds for the primary flags then dest.h=src1.h  
 (もし<cond>が一次フラグのためであれば)  
 else dest.h=src2.h,  
 If<cond>fails for the secondary flags then dest.l=src1.l  
 (もし<cond>が二次フラグのためでなければ)  
 else dest.l=src2.l.

# 11 予約済

ニューモニック :

- 00 SEL<cond> <dest>, <src1>, <src2>  
 01 SELTT<cond> <dest>, <src1>, <src2>  
 10 SELTF<cond> <dest>, <src1>, <src2>  
 11 未使用

レジスタが再充填の印になっていると、それは、無条件で再充填される。また、アセンブラ、次のニューモニックも提供する。

- MOV<cond> <dest>, <src1>  
 SELFT<cond> <dest>, <src1>, <src2>  
 SELFF <cond><dest>, <src1>, <src2>

MOV<cond> A, B は、SEL<cond> A, B, A と等価である。SELFT と SELFF は、SELTF, SELTT を使用して、src1 と src2 を交換す

ることによって得ることができる。

## フラグ

すべてのフラグは、一連の選択が行われるよう保存される。

含む理由：

簡単な決定をブランチに頼ることないインラインにするために使用される。最大要素のためにサンプル又はベクトルをスキャンする時に、そしてビタビ(Viterbi)アルゴリズムによって使用される。

シフト操作命令は、指定量の左右の論理シフト、右算術シフト、回転(rotate)を提供する。シフト量は、レジスタの内容の下8ビットから取られた-128から+127の間の符号付き整数、又は、+1から+31の範囲内の即値である。負の量のシフトは、ABS(シフト量)分反対方向にシフトさせる。

入力オペランドは、32ビットに符号拡張され、結果の32ビット出力は、書き戻し前に48ビットに符号拡張され、48ビットレジスタへの書き込みが感度よく機能する。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

1	010	OPC	F	S	DEST	SI	R	SRC1	SRC2_SEL
				D			I		

OPCは、命令のタイプを指定する。

動作(OPC)：

- 00 dest = (src2>=0) ? src1 <<src2 : src1 >>-src2
- 01 dest = (src2>=0) ? src1 >>src2 : src1 <<-src2
- 10 dest = (src2>=0) ? src1 - >>src2 : src1 <<-src2
- 11 dest = (src2>=0) ? src1 ROR src2 : src1 ROL -src2

ニューモニック：

00 ASL <dest>, <src1>, <src2\_16>  
01 LSR <dest>, <src1>, <src2\_16>  
10 ASR <dest>, <src1>, <src2\_16>  
11 ROR <dest>, <src1>, <src2\_16>

#### フラグ

Zがセットされるのは、結果が0の時。

Nがセットされるのは、結果が負の時。

Vは保存される。

Cは、最後にシフトされて出た（ARM上として）ビット値にセットされる。

レジスタ指定されたシフトの挙動は以下の通り。

- 32によるLSLの結果は0で、src1のビット0にCがセットされる。
- 32を越えるもののLSLは、結果が0で、Cは0にセットされる。
- 32によるLSRの結果は0で、src1のビット31にCがセットされる。
- 32を越えるもののLSRは、結果が0で、Cは0にセットされる。
- 32以上でのASRの結果は充填され、Cはsrc1のビット31に等しい。
- 32でのRORの結果はsrc1に等しく、Cがsrc1のビット31にセットされる。
- 32を越えるnによるRORは、n-32によるRORと同じ結果とキャリーアウト(carry out)になるので、量が1から32の範囲内になるまで、繰り返し32をnから引く。上記参照。

含む理由：

2のべき乗による乗算／除算。ビット及び領域抽出。シリアル・レジスタ。

未定義の命令が、上記命令セットリストで挙げてある。それらの実行により、ピッコロは、実行を停止し、状態レジスタにUビットをセットし、それ自身をディスエーブルする（制御レジスタ内のEビットがクリアされたかのように）。これにより、命令が将来拡張された場合も、それがトラップされて、オプションに、既存の手段上でエミュレートされることが可能である。

ARMからピッコロ状態へのアクセスは以下の通り。状態アクセス・モードを

使用して、ピッコロの状態を観察／変更する。このメカニズムが提供されるのは次の2つの理由からである。

－ 文脈(Context)切替え

－ デバッグ

ピッコロは、PSTATE命令を行うことで、状態アクセスモードになる。このモードでは、ピッコロの状態を退避して、一連のSTC及びLDC命令で復元

される。状態アクセスモードに入ると、ピッコロ・コプロセッサID PICCOL01の使用が変更されて、ピッコロの状態にアクセスできるようになる。ピッコロの状態には7つのバンクがある。特定バンク内のすべてのデータは、単一のLDC又はSTCでロードし記憶することができる。

バンク0：プライベート・レジスタ

- － ピッコロIDレジスタ(Read Only)の値を含む1つの32ビットワード
- － 制御レジスタの状態を含む1つの32ビットワード
- － 状態レジスタの状態を含む1つの32ビットワード
- － プログラム・カウンタの状態を含む1つの32ビットワード

バンク1：汎用レジスタ (GPR)

- － 汎用レジスタの状態を含む16個の32ビットワード

バンク2：アキュムレータ

- － アキュムレータ・レジスタの上の32ビットを含む4つの32ビットワード（注：GPR状態の複製が復元に必要だということは、さもないとレジスタバンク上で別の書き込みイネーブルを意味する）。

バンク3：レジスタ／ピッコロROB／出力FIFO状態

- － どのレジスタが再充填用の印（各32ビットレジスタにつき2ビット）になっているかを示す32ビットワードが1つ。
- － ROBタグ（ビット7から0に記憶されている7ビット項目8つ）の状態を含む32ビットワード8つ。
- － 連合していない(unaligned)ROBラッチ（ビット17から0）の状態を含む32ビットワード3つ。

ー 出力シフトレジスタ内のどのスロットが有効データを含むかを示す32ビットワードが1つ（ビット4は空きを示し、ビット3から0は、使用中のエントリ数をエンコードする）。

ー ラッチ（ビット17から0）を保持する出力FIFOの状態を含む32ビットワード1つ。

バンク4：ROB入力データ

ー 32ビットデータ値が8つ。

バンク5：出力FIFOデータ

ー 32ビットデータ値が8つ。

バンク6：ループハードウェア

ー ループ開始アドレスを含む32ビットワード4つ。

ー ループ最終アドレスを含む32ビットワード4つ。

ー ループ回数（ビット15から0）を含む32ビットワード4つ。

ー ユーザ定義リマッピング・パラメータその他のリマッピング状態を含む32ビットワードが1つ。

LDC命令は、ピッコロが状態アクセスモードにある時にピッコロの状態をロードするのに使う。BANK領域はロードされるバンクを特定する。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	110	P	U	0	W	1	BASE	BANK	PICCOLO1	OFFSET
------	-----	---	---	---	---	---	------	------	----------	--------

次の一連の動作により、ピッコロのすべての状態がレジスタR0内のアドレスからロードされる。

LDP B0, [R0], #16! ; プライベート・レジスタ

LDP B1, [R0], #64! ; 汎用レジスタをロードする。

LDP B2, [R0], #16! ; アキュムレータをロードする。

LDP B3, [R0], #56! ; レジスタ/ROB/FIFO状態をロードする。

LDP B4, [R0], #32! ; ROBデータをロードする。

L D P B 5 , [ R 0 ] , # 3 2 ! ; 出力 F I F O をロードする。

L D P B 6 , [ R 0 ] , # 5 2 ! ; ループハードウェアをロードする。

S T C 命令は、ピッコロが状態アクセスモードにある時にピッコロの状態を記憶させるのに使う。B A N K 領域はどのバンクが記憶されるかを特定する。

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

COND	110	P	U	0	W	0	BASE	BANK	PICCOLO1	OFFSET
------	-----	---	---	---	---	---	------	------	----------	--------

次の一連の動作により、ピッコロのすべての状態がレジスタ R 0 内のアドレス

から記憶される。

S T P B 0 , [ R 0 ] , # 1 6 ! ; プライベートレジスタを退避する。

S T P B 1 , [ R 0 ] , # 6 4 ! ; 汎用レジスタを退避する。

S T P B 2 , [ R 0 ] , # 1 6 ! ; アキュムレータを退避する。

S T P B 3 , [ R 0 ] , # 5 6 ! ; レジスタ / R O B / F I F O 状態を退避する。

S T P B 4 , [ R 0 ] , # 3 2 ! ; R O B データを退避する。

S T P B 5 , [ R 0 ] , # 3 2 ! ; 出力 F I F O を退避する。

S T P B 6 , [ R 0 ] , # 5 2 ! ; ループハードウェアを退避する。

デバッグ・モード - ピッコロは、A R M によってサポートされているものと同じデバッグ・メカニズム、すなわち、DemonとAngelを介したソフトウェア、及び埋め込まれた I C E を備えたハードウェア、に応答しなければならない。ピッコロのシステムをデバッグするためのいくつかのメカニズムがある。

- A R M 命令ブレークポイント

- データ・ブレークポイント (ウオッチポイント)

- ピッコロ命令ブレークポイント

- ピッコロ・ソフトウェア・ブレークポイント

A R M 命令ブレークポイント及びデータ・ブレークポイントは、A R M 埋め込み I C E モジュールによって扱われる。ピッコロ命令ブレークポイントは、ピッ



コロ埋め込みICEモジュールによって扱われる。ピッコロ・ソフトウェア・ブレークポイントは、ピッコロ・コアによって扱われる。ハードウェア・ブレークポイント・システムは、ARMとピッコロの両方がブレークポイントされるように構成される。

ソフトウェア・ブレークポイントを扱うのは、ピッコロ命令 (Halt又はBreak) で、ピッコロに実行を止めさせ、デバッグ・モードに入れ (状態レジスタのBビットがセットされる)、自身をディスエーブルする (ピッコロがPDISABLE命令によってディスエーブルされたようになる)。プログラム・カウンタは有効のままで、ブレークポイントのアドレスが回復できる。ピッコロは、それ以上、命令を実行しなくなる。

Single stepping Piccoloは、ピッコロ命令ストリーム上に次々にブレークポイントをセットすることによって行われる。

ソフトウェア・デバッグ — ピッコロによって提供される基本的機能は、状態アクセスモードにある時、コプロセッサ命令を介して、すべての状態をメモリーにロード及び退避させる能力である。これにより、デバッガーは、すべての状態をメモリーに退避させ、それを読み出し、及び／又は更新し、それをピッコロに復元することができる。ピッコロの記憶状態メカニズムは、非破壊的であり、つまり、ピッコロの状態を記憶する動作は、ピッコロの内部状態を駄目にするものではない。つまり、ピッコロは、その状態をダンプした後、それを復元することなしに、再開できる。

ピッコロ・キャッシュの状態を見つけるメカニズムを決定しなければならない。

ハードウェア・デバッグ — ハードウェア・デバッグは、ピッコロのコプロセッサ・インターフェース上のスキャン・チェインによって行うことができる。ピッコロは状態アクセスモードになり、スキャン・チェインを介して、その状態を調査／変更してもらう。

ピッコロの状態レジスタは、ブレークポイント付き命令を実行したことを示す単一ビットを含む。ブレークポイント付き命令が実行されると、ピッコロは、状

態レジスタにBビットをセットし、実行を中止する。ピッコロに質問をするには、デバッガーは、ピッコロをイネーブルし、次のアクセスが起きる前に、制御レジスタに書き込むことによって、状態アクセスモードにしなければならない。

図4は、Hi/L0ビットとSizeビットに応答して、選択されたレジスタの適当な半分をピッコロ・データパスに切り換えるマルチプレクサ構成を示す。Sizeビットが16ビットなら、符号拡張回路が必要に応じてデータパスの高次ビットに0か1を入れる。

## セクション2

図8は、図1のシステムを変更し発展させたものを示す。中央処理装置（以下、CPUとする）コア2は、命令デコーダ80と、命令パイプライン82と、レジスタバンク84と、メモリコントローラ86とを備える。作動中、CPU命令は、メモリコントローラ86によってメモリ8から引き出され、命令パイプライン8

2に供給される。命令は、命令パイプラインに沿って、命令デコーダ80に隣接する段に到達する。この段において、命令を実行するためのデコーディングは完結する。命令デコーダ80は、命令内のビットフィールドに응答し、CPUコア2の他の部分を構成し駆動する制御信号を提供する論理回路を使用して、所望のデータ処理動作を行う。実際には、CPUコア2は、多数の機能ブロック、例えば、演算ユニット、マルチプライア、キャッシュ、メモリ管理ユニットを備える。

本例では、コプロセッサメモリアクセス命令が命令デコーダ80によってデコードされる時、メモリ8内のメモリ位置を指すアドレスポインタとなるアドレス値を保持するレジスタバンク84内のレジスタが指定される。このアドレス値は、アドレスバスに出されメモリ8に到達し、メモリコントローラを介して、バーストモード転送を開始する。転送されるデータは、CPUコア2よりも、むしろコプロセッサ4に向けられる。従って、メモリ8に対して適切なアドレスを生成する以外は、CPUコア2は、メモリ8とコプロセッサ4との間のデータバス上にあるデータワードに응答しない。命令デコーダ80も、コプロセッサ制御信号

( C P   C o n t r o l ) を生成し、それらはコプロセッサ4に渡される。これらのコプロセッサ制御信号は、コプロセッサに対して、コプロセッサメモリアクセス命令が実行中であることを示し、従って、コプロセッサ4は、データをデータバスに出すか、データバスからデータを読み取るか、適切な動作を行わなければならない。コプロセッサ4に渡されるコプロセッサ制御信号は、実行中のコプロセッサメモリアクセス命令内のアドレッシングモード情報の少なくとも一部を含む。より詳しくは、少なくとも、Pフラグ、Uフラグ、及びオフセット値Mがコプロセッサ4に渡される。

コプロセッサ4は、排他的論理和 ( E O R ) 演算をこれらのビットに行うことによって、PフラグとUフラグをデコードする。この排他的論理和演算の結果によって、コプロセッサは、現在のバーストで転送されるべきデータワードの個数が、レジスタ88に記憶されていて渡されるオフセット値Mによって指定されるか、1つのデータワードというデフォルト値にすべきかを決定する。コプロセッサ転送コントローラ90は、レジスタ88の出力と排他的論理和演算の結果に回答して、データバス上で受け取られたデータワードの個数を数え、指定された数

が受け取られ、バースト終了信号が出ると、それは、メモリ8とCPUコア2に返されて、実行中のコプロセッサメモリアクセス命令によって開始された転送を終了する。コプロセッサ4によって、データバスから受け取られたデータワードは、どれも、リオーダ ( r e o r d e r ) バッファ12にロードされてから、コプロセッサコア92によって処理される。あるいは、コプロセッサ4は、バーストの長さを直接メモリ8へ提供することもできる (これは、例えば、同期D R A Mのような型に有効である)。

図9は、上述のコプロセッサメモリアクセス命令の動作を模式的に示すものである。

動作は、ステップ94で開始され、ステップ96に移り、そこでCPUがコプロセッサメモリアクセス命令に埋め込まれたアドレッシングモード情報を読み取り、一方、コプロセッサは、この同じアドレッシングモード情報の少なくとも一部を読み取り、そこから、転送されるデータワードの個数を決定する。

ステップ 98 で、CPU は、メモリ 8 に提供されるアクセス開始アドレスを生成する。ステップ 100 で、メモリ 8 とコプロセッサ 4 との間で直接的に、データワードが転送される。データワードが 1 つ転送されるごとに、ステップ 102 において、コプロセッサメモリアクセス命令により指定された個数のデータワードがすべて転送されたかどうかを、コプロセッサ 4 が決定する。

転送が完了しないうちは、CPU コア 2 の動作が継続し、ステップ 104 におけるアドレスに必要な更新を行い、ステップ 100 に戻る。

転送が完了すると、システムはステップ 106 に進み、そこで、コプロセッサは、転送を終了するために、バースト完了信号を出して、それがメモリ 8 と CPU コア 2 の両方に渡される。ステップ 108 において、CPU コア 2 は、コプロセッサメモリアクセス命令によって指定されたように、アドレスポインタを更新する（これは、プロセスの他の点において行ってもよい）。このプロセスは、ステップ 110 で終了する。

性能を改善するために、転送は、並行的に行ってもよい。例えば、コプロセッサが CPU に対して、第 1 のワードの転送が開始する前でも、第 2 のデータワードが転送されるべきであるかどうかを指示することもできる。

以下に、本発明の実施の形態により動作する様々のコプロセッサメモリアクセス命令について詳細に述べる。転送されるデータワードの個数を制御するという視点から見ると、これらの命令の全体的な動作は、P フラグと U フラグに従って、次のアドレス転送モードのどれかになる。

	転送開始 アドレス値	アドレスレジスタ 内の最終値	転送される データワード個数
(i)	$R_n$	$R_n - (WL * M)$	1
(ii)	$R_n$	$R_n$	M
(iii)	$R_n$	$R_n + (WL * M)$	M
(iv)	$R_n - (WL * M)$	$R_n$	M
(v)	$R_n - (WL * M)$	$R_n - (WL * M)$	M
(vi)	$R_n + (WL * M)$	$R_n$	1
(vii)	$R_n + (WL * M)$	$R_n + (WL * M)$	1

この他に、ベースレジスタが ARM プログラムカウンタレジスタ (PC 又は R15) である場合の転送されるワードの個数がある。この場合、PEOR (U 又は (ベースレジスタが PC である。)) への単一ワード転送を決定するロジックを変更することになる。

メモリからリオーダバッファへのローディング

フォーマットの要約

リオーダバッファにメモリからのデータを入れるのに 2 つの主要なフォーマットがある。

— LDP

— LPM

どちらの命令のフォーマットも、ARM LDC 命令としてエンコードされる。LDP 命令クラスは、常に、1 つの 32 ビットのデータワードをメモリから転送

する。LPM 命令クラスは、複数のワードを転送するのに使用することができる。命令ビットパタンのレベルでは、LDP と LPM は、使用されるアドレッシングモードビットによって区別され、アセンブラーの文法は、LDP と LPM に対して異なったニューモニックを使用し、1 つのワードか 2 つ以上のワードの転送か

についてのコードを書く又は読む人に気付かせる。この2つの命令に対して、以下のフォーマットが使用できる。

```

LDP{cond}[32|16]      dest, address
LDP{cond}16U          bank, address
LPM{cond}[A|D][32|16] Rn{!}, [<Rlist>]{, #<wordcount>}
LPM{cond}[A|D]16      Rn{!}, <bank0.1>, #<wordcount>

```

ここにおいて、

{ } は、オプションフィールドを示す。

cond は、ARM命令条件コードフィールドである。

32 | 16 は、ロードされているデータが16ビットデータとして扱われエンディアンネス (endianness) 用アクションを取る (STP16及びSTP32を参照) べきか、あるいは32ビットデータとして扱われるべきかを示す。

dest は、ピッコロ目的レジスタ (A0乃至Z3) を指定する。

address は、以下のいずれかである。

```

[Rn]
[Rn, #+ve__offset] {!}
[Rn] #+ve__offset

```

Rn は、有効なARMレジスタ番号を求める表現である。

! は、計算されたアドレスがベースレジスタに書き戻す必要があることを示す。

#+ve\_\_offset は、 $+<8\text{-bit-offset}> \cdot 4$ として表すことのできるオフセットを求める表現である。このオフセットは、ベースレジスタに加算されロードアドレスを形成する。即ち、アドレスが前もってインデックスをつけられる。

#-ve\_\_offset は、 $-<8\text{-bit-offset}> \cdot 4$ として表すことのできるオフセットを求める表現である。このロードアドレスは、ベースレジスタRnの値であり、オフセット分がRnから減算され、結果がRnに書き戻

される。

`bank` は、3つの非蓄積型ピッコロバンク (`X | Y | Z`) の1つを指定する。

`A | D` は、プレインクリメント (上り)、又は、ポストデクリメント (下り) アドレッシングモードを指定する。

`< R l i s t >` は、ピッコロレジスタ番号の上りリストであり、例えば、`{ X 0 , X 1 }` というように、`{ }` に挟まれている。ピッコロの第1版では、最大8つのレジスタを指定することができる。このリストは、レジスタバンクのトップで巡回しても良い。例えば、`{ Y 2 , Y 3 , A 0 , A 1 }` は、有効なレジスタリストである。

`bank 0 . 1` は、4個の16ビットレジスタ (`A 0 . 1 | X 0 . 1 | Y 0 . 1 | Z 0 . 1`) の1つを指定する。

`word count` は、選択された領域のレジスタ上でラッピング (`w r a p p i n g`) ロードするのに使用され、転送されるデータ項目の総数を指定する。ピッコロの第1版では、1つのLPM命令で最大8つまでのデータ項目を転送することができる。

`< l i s t >` フォーマットが使用される場合、`< l i s t >` によって指定されるレジスタのリストは、(`A 0 , X 0 , Y 0 , Z 0`) のいずれかで始めなければならない。レジスタリストは、1個、2個、又は4個のレジスタを指定することができる。即ち、レジスタのXバンクに対して、

`{ X 0 }`

`{ X 0 , X 1 }`

`{ X 0 , X 1 , X 2 , X 3 }`

だけが、有効な`< R l i s t >`組み合わせである。`< word count >` は、(`R l i s t`) の長さより大きくなければならない。このLPM命令のフォーマットは、`< word count >` データ項目全部を転送する。その際、

`< R l i s t >` 内の各レジスタに対して、リストの最後に来たら、

`< R l i s t >` の開始へ巡回するようにデータにタグをつける。

<bank 0. 1>フォーマットが使用される場合は、ピッコロの第1版では、<wordcount>は、(1-8)の範囲が可能である。このLPM命令のフォーマットは、 $2^{\ast} \text{<wordcount>}$  16ビットのデータ項目をロードし、その際、すべてのデータに、レジスタ<bank 0. 1>用タグをつける。

#### 例

LDPNE 32 A1, [R0] : Zフラグ=0であれば、A1にmem(R0)をロードする。データを32ビットとして扱う。

LDP 16 X0, [R0, #16] ! : X0にmem(R0+16 (バイト))をロードし、R0+16をR0に書き戻す。データをパケット(pack ed) 16ビットとして扱う。

LDP 32 Y2, [R0], #-4 : Y2にmem(R0)をロードし、R0に(R0-#4)を書き戻す。

LDP 16 U X, [R0, #4] : Xバンクの非配列ラッチにmem(R0+#4 (バイト))を入れる。

LPMEQA 32 R0!, {X2, X3, Y0, Y1} : Zフラグ=1であれば、4ワードを上り順で、メモリ[R0]からロードする。その際、レジスタX2, X3, Y0, Y1用にタグをつける。R0に書き戻す。

LPMA 16 R1!, {X0, X1}, #8 : パケット16ビットデータ8ワードを、上り順で[R1]のメモリからロードする。その際、レジスタX0, X1, X0, X1, X0, X1用にタグをつける。

#### 命令エンコーディング

##### LDP 命令

LDP命令は、1つの32ビットワードをメモリから転送する。これらの命令のいくつかは、書き戻しを行うが、アセンブラの文法に従って、「!」の印がついていない。なぜなら、ポスト・インデクシングは常に書き戻しを含むからである。2つの変形がある。

LDP {cond} [32 | 16] dest, [Rn], #-ve \_



offset

LDP {cond} [32 | 16] dest, [Rn, #-ve\_\_offset] {!}

アドレッシングモードは、P、U、Wビットによって決まる。これらは、それぞれ命令内のビット24、23、21にある。

P = 0, U = 0, W = 1 という組み合わせは、次の形式の命令をエンコードするのに使用される。

LDP {cond} [32 | 16] dest, [Rn], #-ve\_\_offset

アドレス [Rn] からは、ワードが1つだけ転送される。転送が行われた後、ベースレジスタは、8\_\_bit\_\_offset\*4だけデクリメントされる。Nビットは、LDP32(1)またはLDP16(0)を指定する。<8\_\_bit\_\_offset>は、mod(#-ve\_\_offset)/4をエンコードする。この命令は、データ構造を介して逆戻りして1つの特定のワードを出現毎に抽出するのに役立つ。同様の所望の機能は、他の命令とも合わせられる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 0 | 0 0 | N | 1 1 | Rn | dest | pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

P = 1, U = 1 の組み合わせは、次の形式の命令をエンコードするのに使用される。

LDP {cond} [32 | 16] dest, [Rn, #-ve\_\_offset] {!}

アドレス [Rn #-ve\_\_offset] からはワードが1つだけ転送される。Wビットがセットされると、ベースレジスタは、8\_\_bit\_\_offset\*4だけインクリメントされる。即ち、オプションの「!」により書き戻しが指定されている形式である。W = 0 の場合は、書き戻しは行われず、「!」もない。

N

ビットは、LDP32 (1) 又はLDP16 (0) を指定する。< 8 \_ b i t \_ o f f s e t > は、# + v e \_ o f f s e t / 4 をエンコードする。P i c \_ 1 (及び、後で述べるP i c \_ 2) は、コプロセッサがピッコロ・コプロセッサであることを示す、識別番号である。ピッコロは、関連命令に従って使用される2つの識別番号を持っている。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 0 | 1 1 | N | W | 1 | Rn | dest | pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

#### LDP16U

LDP16U命令は、3つの非配列保持ラッチの1つを用意するのに使用される。それは、次の変形を持つ。

```

LDP { cond } U16 bank, [Rn], # - v e _ o f f s e t
LDP { cond } U16 bank, [Rn, # + v e _ o f f s e t] { !
}

```

アドレッシングモードはP及びUビットにより決まる。

P = 0, U = 0, W = 1 という組み合わせは、次の形式の命令をエンコードするのに使用される。

```

LDP { cond } U16 dest, [Rn], # - v e _ o f f s e t

```

アドレス [Rn] から転送されるワードは1つだけである。この転送の後、ベースレジスタは、# - v e \_ o f f s e t により変更される。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 0 | 0 0 | 0 | 1 | 1 | Rn | bnk | 0 0 | pic_2 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

< b n k > は、非配列モードにするバンクを指定し、バンク X, Y, Z に対して

1、2、3の値を取ることができる。

P = 1、U = 1 の組み合わせは、次の形式の命令をエンコードするのに使用される。

```
LDP {Cond} U16 dest, [Rn, #-ve__offset] {!}
}
```

アドレス [Rn + # + ve\_\_offset] から転送されるワードは1つだけである。Wビットがセットされると、ベースレジスタは、# + ve\_\_offset により変更される。W = 0 なら、書き戻しはない。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond |1 1 0|1 1|0|W|1| Rn |bnk|0 0| pic_2 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

< b n k > は、非配列モードにするバンクを指定し、バンク X, Y, Z に対して、1、2、3 の値を取ることができる。

LPM 命令

LPM 命令は、メモリから2つ以上のワードを転送し、次の変形がある。

```

LPM{cond}[A|D]{32|16} Rn{!}, [<Rlist>]
LPM{cond}[A|D]{32|16} Rn{!}, [<Rlist>], #<wordcount>
LPM{cond}[A|D]16 Rn{!}, <bank0.1>, #<wordcount>

```

LPMA 形に対しては P = 0、U = 1 であり、命令は、次のようにエンコードされる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond |1 1 0|0 1|N|W|1| Rn |basereg| pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

ここにおいて、

Nビットは、LPMA32 (1) 又はLPMA16 (0) を指定する。

Wビットは、W = 1 なら、ベースレジスタへの basereg = offset

\* 4 の書き戻しを指定する。

< b a s e r e g > は、< R l i s t > 内の第 1 ピッコロレジスタを指定する

。

< 8 \_ b i t \_ o f f s e t > は、転送されるべきレジスタの個数を指定する。  
L P M D 形に対しては P = 1 , U = 0 であり、命令は、次のようにエンコードされる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 0 | 1 0 | N | W | 1 | Rn | { basereg | pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

LPM{cond}[A|D][32|16] Rn{!}, <Rlist>, #<wordcount>

L P M A 形に対しては P = 0 , U = 1 であり、命令（この場合も、以下においても、p i c - 2 , b n k w r p フォーマットを使用する）は、次のようにエンコードされる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 0 | 0 1 | N | W | 1 | Rn | { bnk | wrp | pic_2 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

ここにおいて、

Nビットは、L P M D 3 2 ( 1 ) 又は L P M D 1 6 ( 0 ) を指定する。

Wビットは、W = 1 なら、ベースレジスタへの b a s e r e g + o f f s e t

\* 4 の書き戻しを指定する。

< b n k > は、< R l i s t > 内の第 1 レジスタを指定し、それがバンクのベースでなければならない（即ち、A 0 , X 0 , Y 0 , 又は Z 0 ）。それは、バンク A - Z を指示する値 0 - 3 を取ることができる。

< w r p > は、ラッピング ( w r a p p i n g ) 点を指定し、ラップ値として 1 - 3 の値を取ることができ、それぞれ、2 個、4 個、8 個の 1 6 ビットレジス

タを表す。

< 8 \_ b i t \_ o f f s e t > は、転送すべきデータの個数を指定する。それは、< w o r d c o u n t > / 4 の値を取る。

L P M D 形に対しては  $P = 1$ 、 $U = 0$  であり、命令は、次のようにエンコードされる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond  | 1 1 0 | 1 0 | N | W | 1 |  Rn   | bnk | wrp | pic_2 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

LPM{cond}[A/D]16      Rn{!}, <bank0.1>, #<wordcount>

L P M A 形に対しては  $P = 0$ 、 $U = 1$  であり、命令は、次のようにエンコードされる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond  | 1 1 0 | 0 1 | 0 | W | 1 |  Rn   | bnk | 0 1 | pic_2 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

ここにおいて、

W ビットは、ベースレジスタ (1) 書き戻しを指定する。

< b n k > は、転送先のピッコロバンクを指定し、A, X, Y, 又は Z を表す値 0 - 3 を取ることができる。

< 8 \_ b i t \_ o f f s e t > は、転送すべきデータ項目の個数を指定する。それは、< w o r d c o u n t > / 4 の値を取る。

L P M D 形に対しては  $P = 0$ 、 $U = 1$  であり、命令は、次のようにエンコードされる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 0 | 0 1 | 0 | W | 1 | Rn | bnk | 0 1 | pic_2 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

ARMレジスタからのデータのリオードバッファへのローディング

フォーマットの要約

ARMレジスタからピッコロ・リオードバッファへデータワードを転送するためのMPR命令フォーマットがある。MPR命令には、以下のフォーマットが利用できる。

MPR { cond } dest, Rn

MPR { cond } W dest, Rn

ここで、

{ } は、オプションフィールドを示す。

cond は、ARM命令条件コードフィールドである。

dest は、ピッコロ目的レジスタ (A0-Z3) を指定する。

Rn は、有効なARMレジスタ番号を求める表現である。

W は、ARMレジスタから転送されるデータが、2個の16ビット値として扱われ、ピッコロレジスタdest0、1用のタグをつけなければならないことを示す。

命令コーディング

MPR命令は、ARM MCR命令としてエンコードされる。

**MPR{cond} dest, Rn**

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+
| cond | 1 1 1 0 | 0 0 1 0 | dest | Rn | pic_1 | 0 0 0 | 1 | 0 0 0 0 |
+-----+-----+-----+-----+-----+-----+

```

**MPR{cond}W dest, Rn**

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+
| cond | 1 1 1 0 | 0 0 1 0 | dest | Rn | pic_2 | 0 0 0 | 1 | 0 0 0 0 |
+-----+-----+-----+-----+-----+-----+

```

出力 F I F O からメモリへのデータ記憶

フォーマットの要約

出力 F I F O からメモリへのデータ記憶には、2つの主要命令がある。

— S T P

— S P M

どちらの命令も、ARM S T C 命令としてエンコードされる。S T P 命令クラスは、常に、1つの32ビットのデータワードを出力 F I F O からメモリへ記

憶する。S T M 命令クラスは、2つ以上のワードを記憶するのに使用される。これら2つの命令には、次のフォーマットが使える。

S T P { c o n d } [ 3 2 | 1 6 ] a d d r e s s

S T M { c o n d } [ A | D ] [ 3 2 | 1 6 ] R n { ! } ,

# < w o r d c o u n t >

ここで、

{ } は、オプションフィールドを示す。

c o n d は、A R M命令条件コードフィールドである。

3 2 | 1 6 は、記憶されるデータが 1 6 ビットデータとして扱われ、（前述された）エンディアネス用動作が行われるべきか、3 2 ビットデータとして扱われるべきかを示す。

a d d r e s s は、次のいずれかになる。

[ R n ]

[ R n , # + v e \_ o f f s e t ] { ! }

[ R n ] , # + v e \_ o f f s e t

R n は、有効な A R Mレジスタ番号を求める表現である。

! は、計算されたアドレスがベースレジスタに書き戻されなければならないことを示す。

# + v e \_ o f f s e t は、 $+ < 8 \_ b i t \_ o f f s e t > * 4$ として表すことのできるオフセットを求める表現である。このオフセットは、ベースレジスタに加算されて、記憶アドレスを形成する。

# - v e \_ o f f s e t は、 $- < 8 \_ b i t \_ o f f s e t > * 4$ として表すことのできるオフセットを求める表現である。このオフセットは、ベースレジスタに減算されて、ポストストアアドレスを形成する。

A | D は、プレインクリメント（上り）又はポストデクリメント（下り）アドレッシングモードを示す。

w o r d c o u n t は、転送されるデータの総数項目を示す。ピッコロの第 1 版では、1 つの S P M命令で、最大 8 個のデータ項目を転送できる。

命令エンコーディング

S T P 命令

S T P 命令は、1 個の 3 2 ビットワードをメモリに転送する。2 つの変形がある。

S T P { c o n d } [ 3 2 | 1 6 ] d e s t , [ R n ] , # - v e \_ o f f s e t

S T P { c o n d } [ 3 2 | 1 6 ] d e s t , [ R n , # + v e \_ o f f s e t



t] {!}

アドレッシングモードは、P、Uビットにより決まる。

STP {cond} [32 | 16] [Rn], #-ve\_\_offset (P = 0 | U = 0 | W = 1)

P = 0, U = 0, W = 1 の組み合わせは、次の形式の命令をエンコードするのに使用される。

STP {cond} [32 | 16] [Rn], #-ve\_\_offset

アドレス [Rn] に転送されるワードは、1個だけである。転送が行われた後、ベースレジスタは、8\_\_bit\_\_offset\*4 だけデクリメントされる。

Nビットは、STP 32 (1) 又は STP 16 (0) を指定する。W = 0 のエンコーディングは許可されない。<8\_\_bit\_\_offset> は、mod (#-ve\_\_offset) / 4 をエンコードする。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 0 | 0 0 | N | 1 0 | Rn | 0 0 0 0 | pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

STP{cond}[32|16] dest, [Rn, #+ve\_offset]{!} (P=1|U=1)

P = 1, U = 1 の組み合わせは、次の形式の命令をエンコードするのに使用される。

STP {cond} [32 | 16] dest, [Rn, #+ve\_\_

offset] {!}

アドレス [Rn, #+ve\_\_offset] に転送されるワードは1つだけである。Wビットがセットされると、ベースレジスタは、8\_\_bit\_\_offset\*4 だけインクリメントされる。Nビットは、STP (1) 又は STP 16 (0) を指定する。<8\_\_bit\_\_offset> は、#+ve\_\_offset / 4 をエンコードする。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond  | 1 1 0 | 1 1 | N | W | 0 |  Rn   | 0 0 0 0 | pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

## S P M 命令

S P M 命令は、メモリから2つ以上のワードを転送する。次の変形がある。

SPM{cond}[A|D][32|16] Rn{!}, #<wordcount>

SPM{cond}[A|D][32|16] Rn{!}, #<wordcount>

S P M A に対しては、P = 0, U = 1 であり、命令は次のようにエンコードされる。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond  | 1 1 0 | 0 1 | N | W | 0 |  Rn   | 0 0 0 0 | pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

ここで、

Nビットは、S P M A 3 2 ( 1 ) 又は S P M A 1 6 ( 0 ) を指定する。

Wビットは、ベースレジスタ ( 1 ) への書き戻しを指定する。

< 8 \_ b i t \_ o f f s e t > は、転送すべきデータ項目の数を指定する。

L P M D に対しては、P = 1, U = 0 であり、命令は、次のようにエンコード

される。

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond  | 1 1 0 | 1 0 | N | W | 0 |  Rn   | 0 0 0 0 | pic_1 | 8_bit_offset |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

ここで、

Nビットは、S P M D 3 2 ( 1 ) 又は S P M D 1 6 ( 0 ) を指定する。

出力FIFOからARMへのデータ転送

フォーマットの要約

M P R 命令フォーマットは、出力FIFOからARMレジスタへデータワードを転送するためにある。M P R 命令には、次のフォーマットがある。

M P R { c o n d } R n

ここで、

{ } は、オプションフィールドを指示する。

c o n d は、ARM命令条件コードフィールドである。

R n は、有効なARMレジスタ番号を求める表現である。

命令エンコーディング

M R P 命令は、ARM M R C 命令としてエンコードされる。

MRP{cond} Rn

```

3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
3 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
+-----+-----+-----+-----+-----+-----+-----+-----+
| cond | 1 1 1 0 | 0 0 1 1 | dest | Rn | pic_1 | 0 0 0 1 | 0 0 0 0 |
+-----+-----+-----+-----+-----+-----+-----+-----+

```

予備オプション

全体的なレベルとして、P = 0 , U = 0 , W = 0 は、以上の形式では許可されない。将来必要になった時の予備である。

【 図 1 】

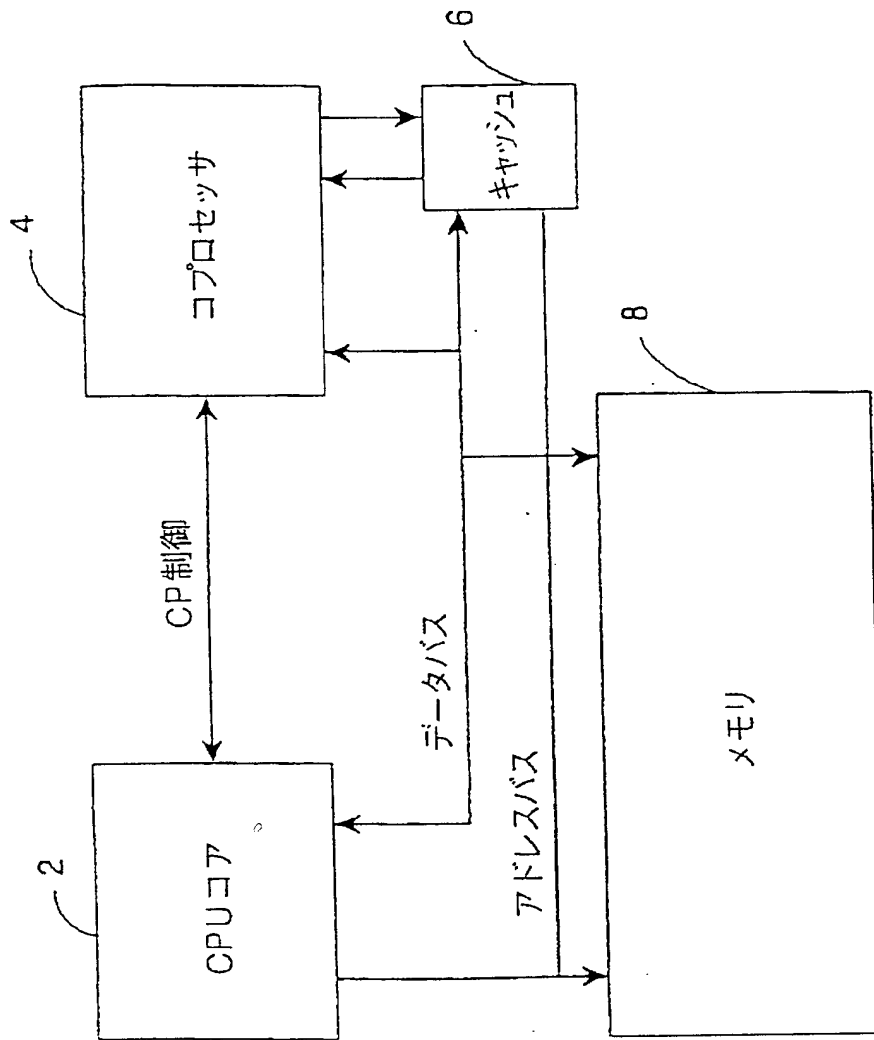


Fig. 1

【 図 2 】

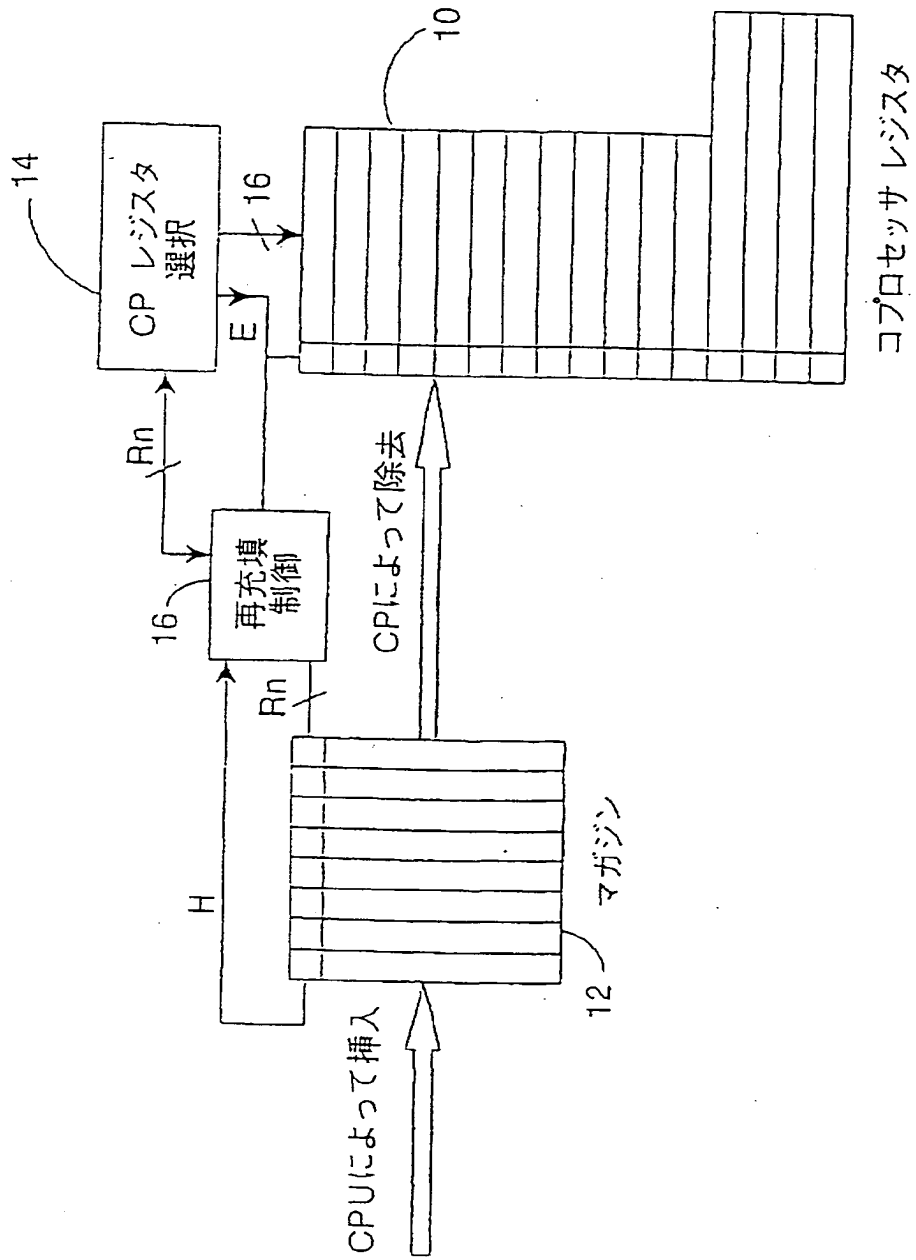


Fig.2

【 図 3 】

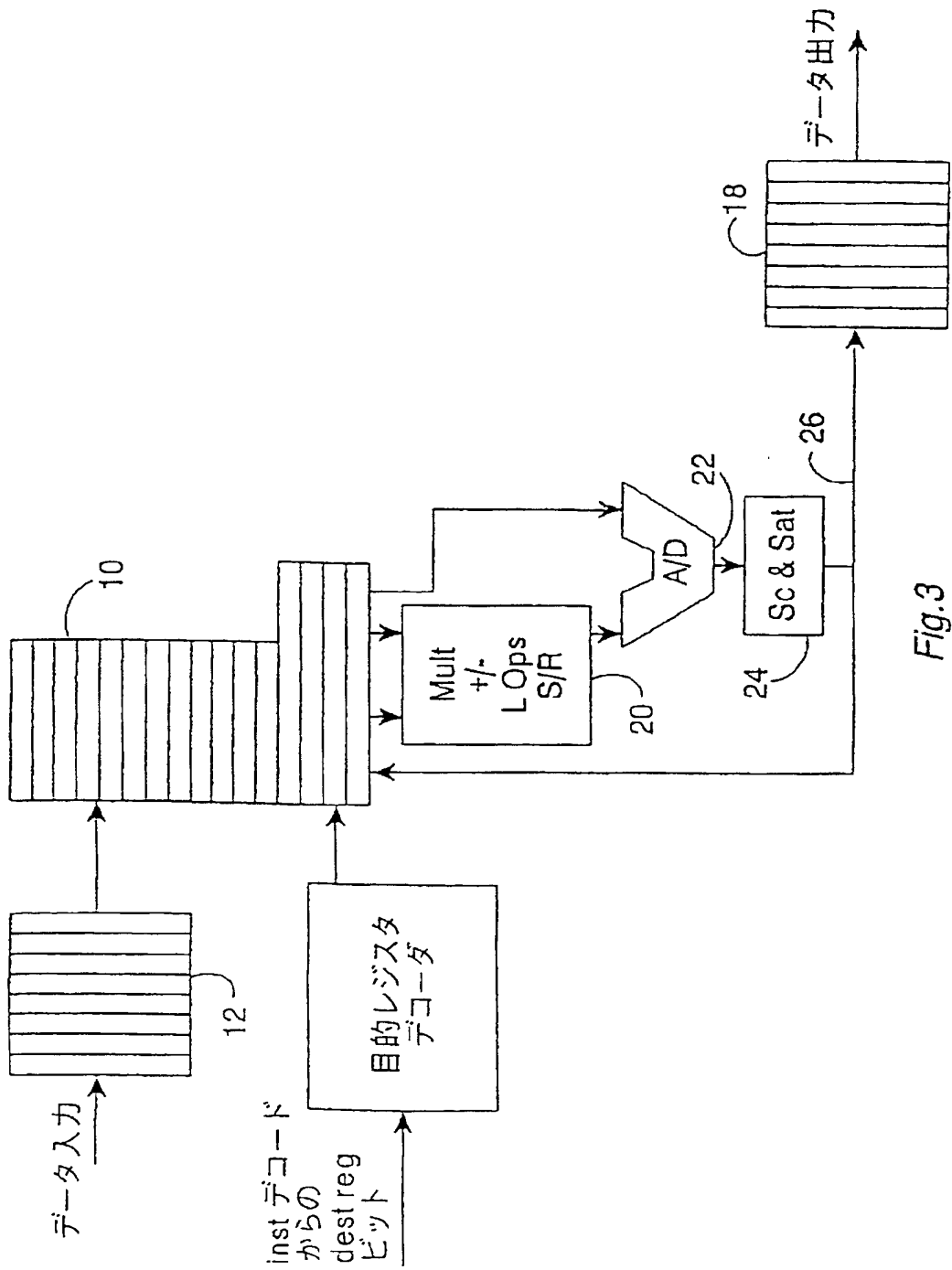


Fig.3

【 図 4 】

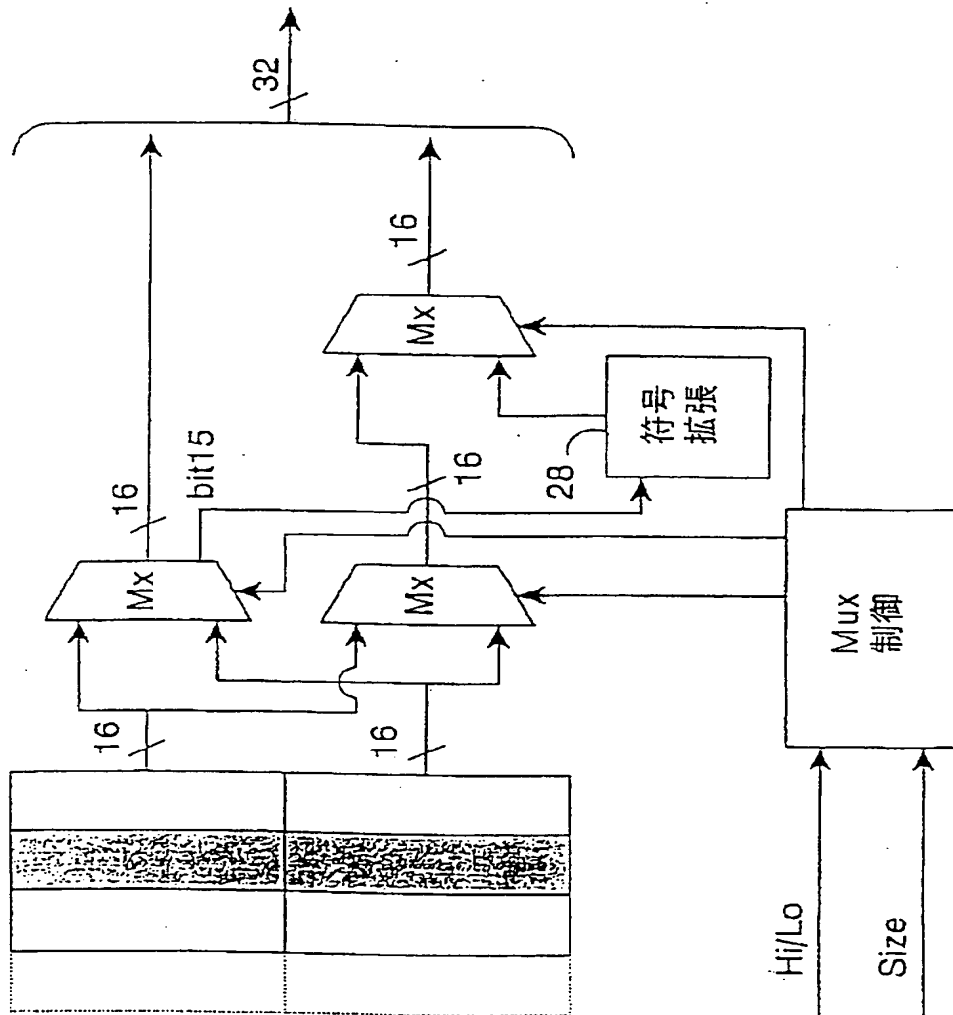


Fig.4

【 図 5 】

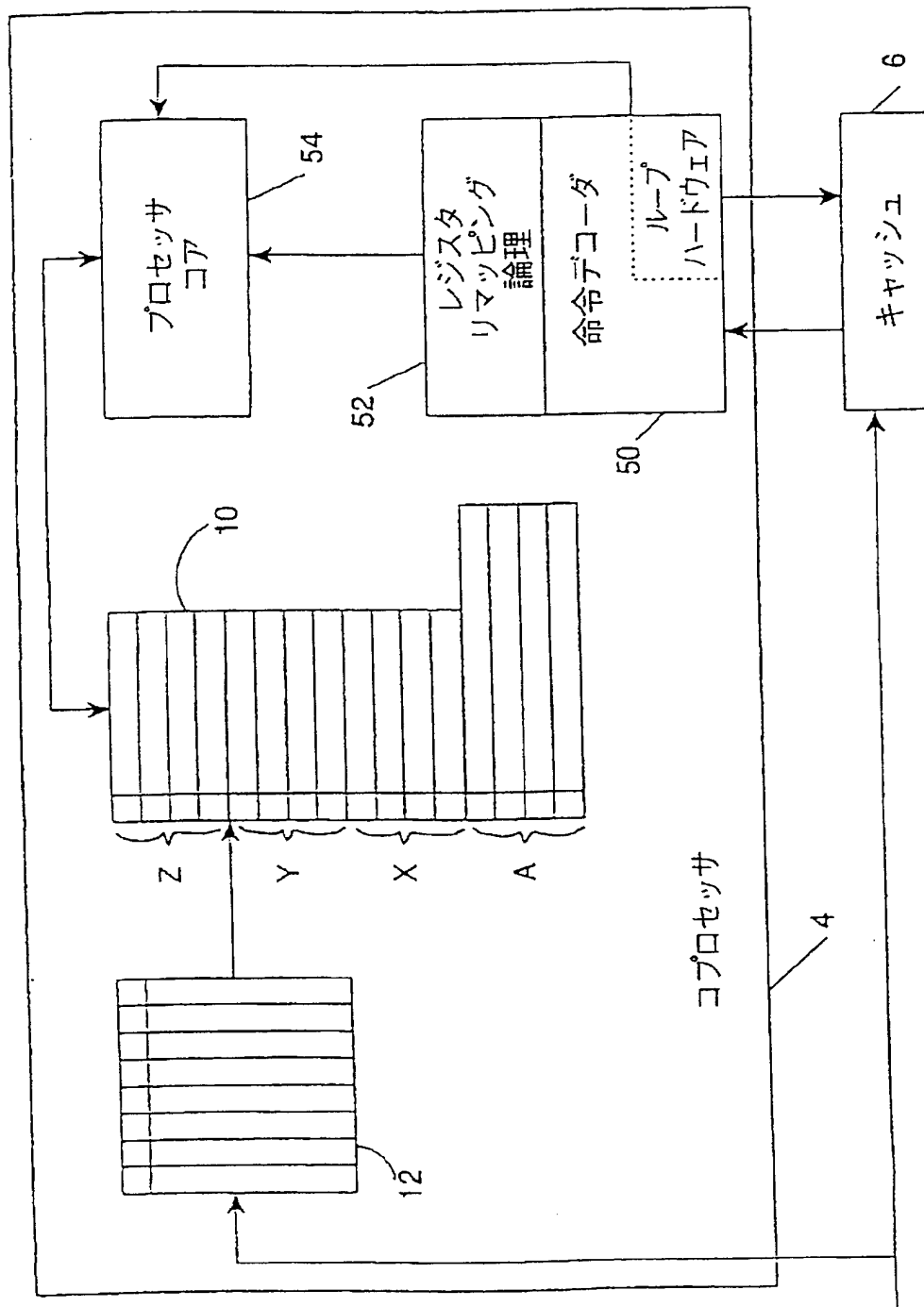
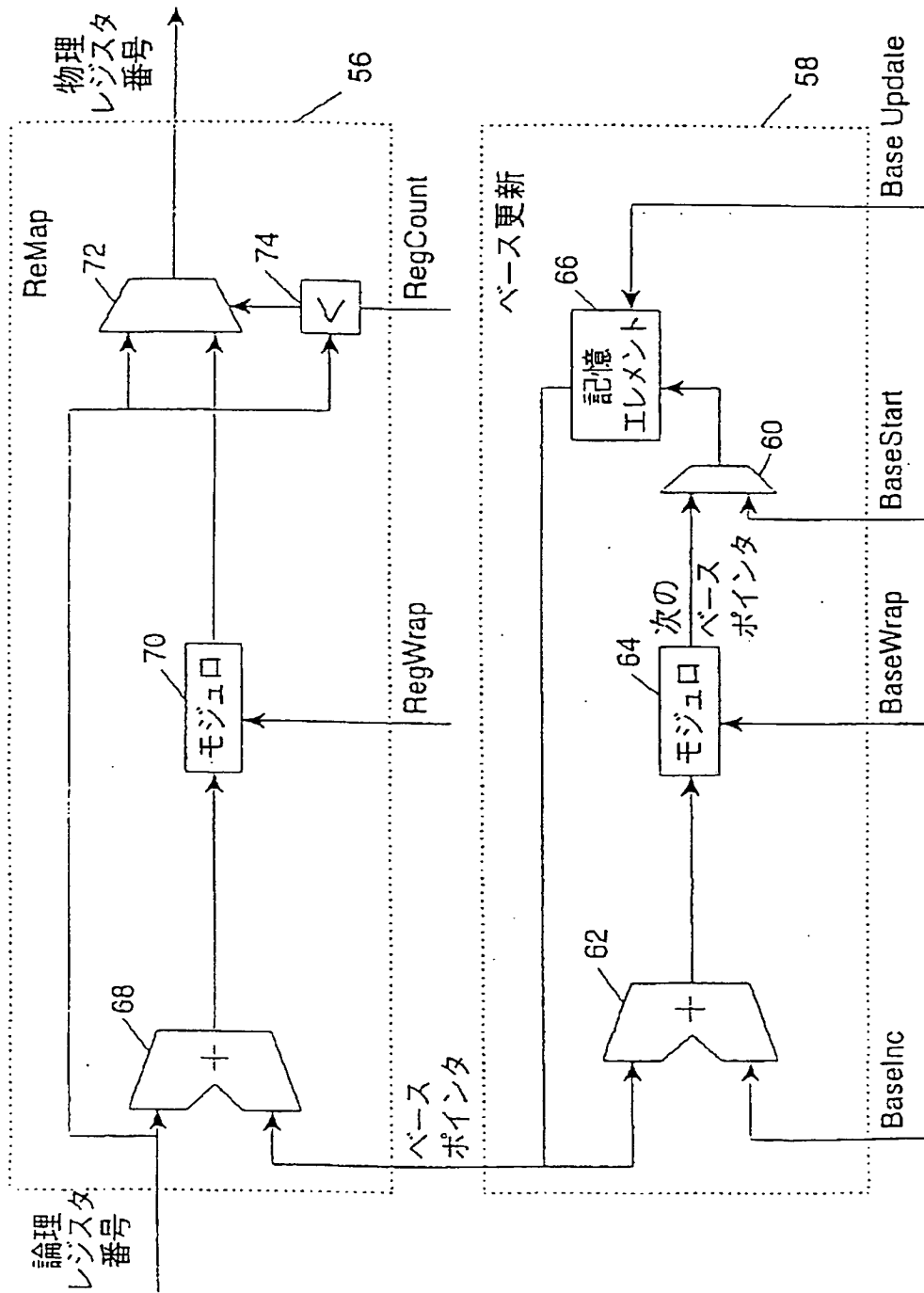


Fig.5



【 図 6 】



【 図 7 】

FIR (ブロック・フィルタ・アルゴリズム)

	d0	d1	d2	d3	d4	d5
A0=	c0	c1	c2	c3	c4	c5
A1=		c0	c1	c2	c3	c4
A2=			c0	c1	c2	c3
A3=				c0	c1	c2

Fig.7

【 図 8 】

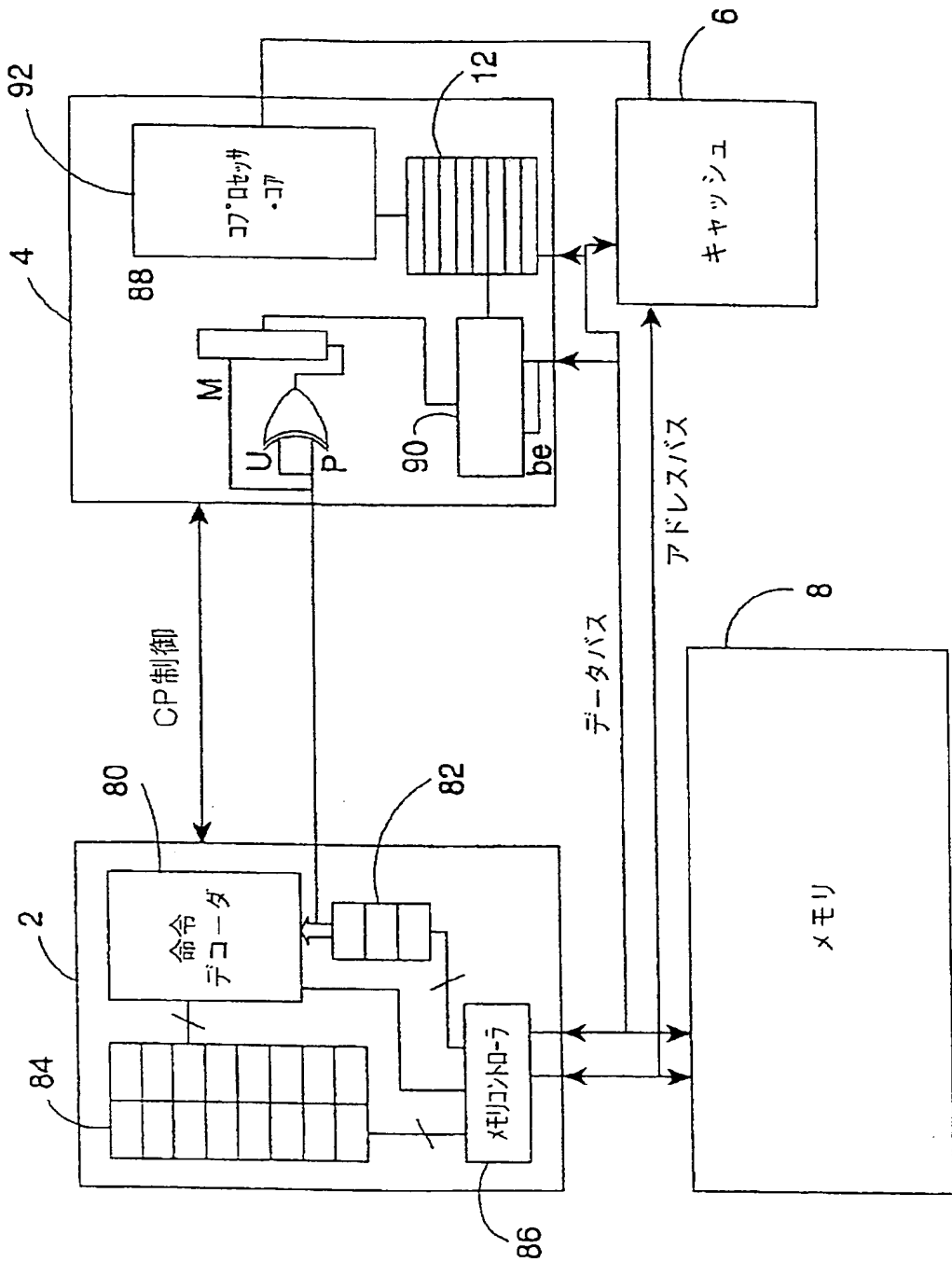


Fig.8

【 図 9 】

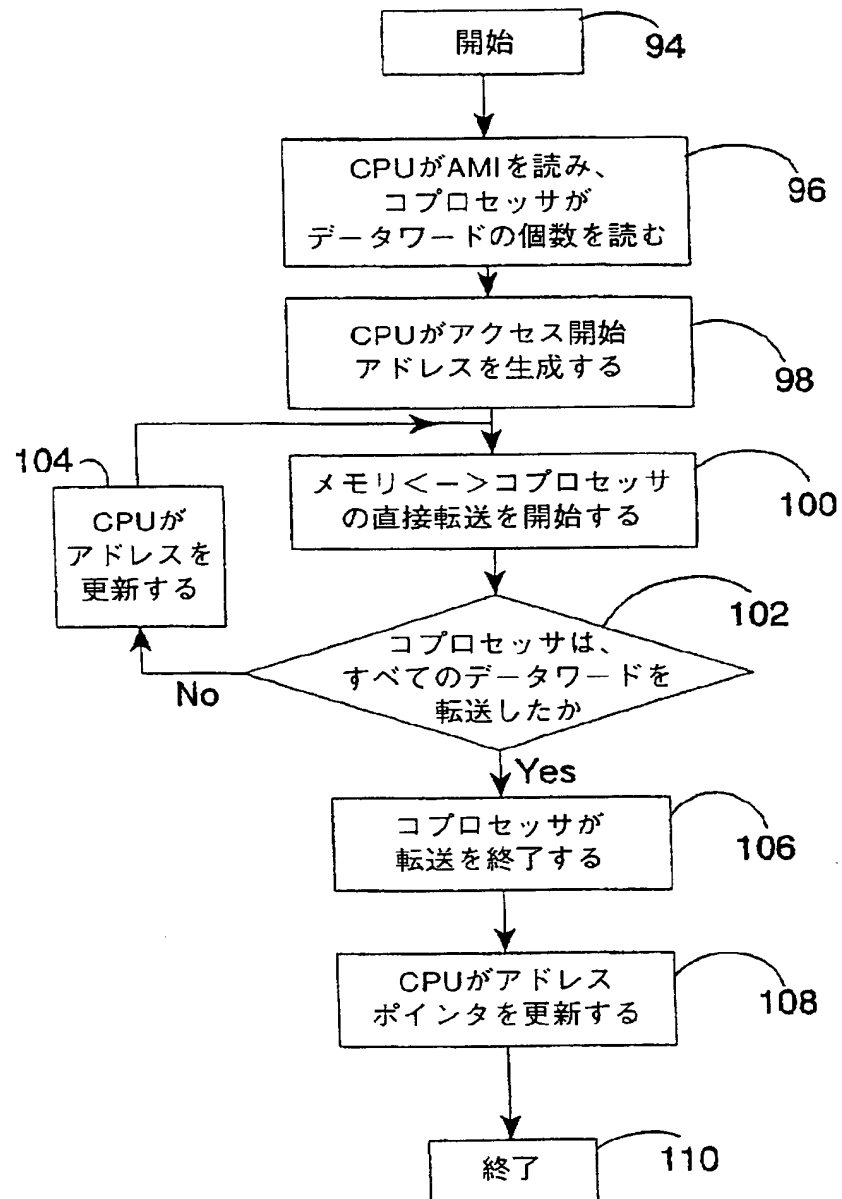


Fig.9

【手続補正書】特許法第 184 条の 8 第 1 項

【提出日】平成 11 年 4 月 21 日 ( 1999 . 4 . 21 )

【補正内容】

(例えば、ARM 浮動小数点アクセラレータユニットにおけるように)、中央処理装置上で実行する命令内にビットフィールドを割り当てることが公知であり、そこでは、ビットフィールドがコプロセッサに渡され、転送されるべきデータワードの個数をコプロセッサに指定する。しかしながら中央処理装置上で実行される命令内で使えるビットスペースには限界があり、もし、命令内のビットがコプロセッサへのデータワードの個数を渡すことに専念すると、命令内の他のフィールド用に使えるビットスペースを制限することになる。それは例えば、命令実行に続く中央処理装置内のアドレスポインタの変更のような、データ転送に関する他パラメータを指定するのに使用されるかもしれない。

S B ファーバー (Furber) 著「VLSI RISC アーキテクチャ及び構築 (VLSI RISC Architecture and Organization)」1989、マルセル・デッカー (Marcel Dekker) 株式会社、ニューヨーク XP002061358201970、ページ 261-265 は、中央処理装置により使用されるアドレッシングモードを制御するビットと、転送されるデータワードの個数を制御するためにコプロセッサにより使用される別のビットとを備えたコプロセッサ命令を開示している。

欧州公開特許出願 EP-A-0, 703, 529 は、あるオペコードによって、オフセットフィールドの一部が、使用されるオフセットよりむしろ転送されるワード数を制御するメモリアクセス命令を備えたマイクロプロセッサを開示している。

本発明の一面は、データ処理用装置であって、

コプロセッサメモリアクセス命令を含むデータ処理動作を行う中央処理装置命令を実行する中央処理装置と、

この中央処理装置に結合され、データワードを保持するメモリと、

中央処理装置とメモリに結合されたコプロセッサであって、このコプロセッサにより処理されるべきメモリ内のデータワードの指定が、中央処理装置により実

行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレッシングモードの1つを使用して行われることを特徴とするコプロセッサと

を備え、

少なくとも1つのコプロセッサメモリアクセス命令が、前記複数のアドレッシングモードのどれが中央処理装置によって使用されるかを制御するアドレッシングモード情報を含み、

コプロセッサは、前記アドレッシングモード情報の少なくとも一部を使用して、前記少なくとも1つのコプロセッサメモリアクセス命令に応答してメモリとコプロセッサの間で何個のデータワードが転送させるべきかを制御することを特徴とする。

本発明は、中央処理装置が複数のアドレッシングモードのどれを使用するかを制御するのに使用する（レジスタ番号や即値定数を含むことのある）ビットフィールドは、コプロセッサに、転送されるべきデータワードの個数を指定するのにも使用される（命令内の他のフィールドあるいは制御レジスタに書き込まれた値のような他の因子と結合されている可能性がある）ことがある。例えば、これまでにわかっているところ、多くの場合、中央処理装置が、転送及び／又はアドレスポインタへの変更に使用されアドレスを制御するのに使用するビットフィールド情報は、コプロセッサに転送されるデータワードの個数と関係づけられるので、このビットフィールドは、コプロセッサによっても中央処理装置によっても読むことができる。コプロセッサメモリアクセス命令内で同じビットフィールドを重複して使用することにより、このようなコプロセッサメモリアクセス命令内のビットスペースを他の目的に開放することになる。更に、現在わかっているところでは、大多数の場合、転送されるべきデータワードは、少数のカテゴリに分類されるので、コプロセッサに転送されるべき個数のデータワード用に専用ビットフィールドを提供するフルフレキシビリティは、本発明を使用することによって得られるコード密度と性能の改善に負ける。

中央処理装置は、いくつかの異なる方法でアドレッシングを制御できるのに対して、本発明の好ましい実施の形態においては、前記少なくとも1つのコプロセッサ

サメモリアクセス命令は、アドレス値を保持する中央処理装置内のレジスタを参照し、前記アドレスモード情報は、オフセットフィールドを備え、アクセスされるべきメモリ内の開始アドレスの決定は、前記少なくとも1つのコプロセッサメモリアクセス命令の実行により、前記アドレス値と前記オフセット値の少なくとも1つから行われる。

中央処理装置のレジスタ内に保持されるアドレスポインタのそのような使用は、多大なフレキシビリティを提供し、命令内のオフセットと結合されて、アプリケーションのコーディングを簡単にする。

コプロセッサとデータアレイを介して動作することが所望される場合、本発明の好ましい実施の形態では、前記アドレス値に行われた変更が、最終アドレス値を生成し、それを前記レジスタへ記憶しなおすことができるようにする。

前記オフセットフィールドの少なくとも一部が前記コプロセッサに使用されて、データワードが何個、前記メモリと前記コプロセッサの間で転送されるかを制御できることは、非常に便利であることがわかった。

このような装置は、メモリ内に保持されているデータ上にデータ処理操作をするためにコプロセッサが使用される実際の状況のかなりの部分の要求を満たすことができる。

また、アドレッシングモード情報が1つ以上のフラグを備え、前記複数のアドレッシングモードのうちのどれが使用されるか、また、何個のデータワードが前記メモリと前記コプロセッサの間で転送されるかを決定するのに前記オフセットフィールドが使用されるか否かを制御できることは有利である。

アドレッシングモード情報内の他のフラグビットに依存してオフセットフィールドを選択的に使用することによって、転送されるデータワードの個数をコプロセッサが制御できる方法のオプションの数を増加させ、それにより、コプロセッサメモリアクセス命令内のビット空間を余分に必要とすることなしに、実際の状況の、より高い部分の要求に合わせることができる。

高い割合の所望のタイプの動作を成就するには、好ましい実施の形態は、次のようなものでなければならない。即ち、前記コプロセッサが、前記オフセットを

使用せずに、前記メモリと前記コプロセッサの間で転送されたデータワードの個数を決定する場合、メモリとコプロセッサの間では、固定数のワードが転送される。

コプロセッサを制御するコプロセッサメモリアクセス命令モードの相補セットは、次のようなものである。即ち、前記レジスタがアドレス  $R_n$  を記憶し、デー

タワードが  $WL$  バイトの長さで、前記オフセット値が  $M$  であり、前記 1 つ以上のフラグが、3 つ以上の値ビットを備え、それらが、前記少なくとも 1 つのコプロセッサアクセス命令を選択し、以下の 1 つに従って操作する。

	転送開始 アドレス値	アドレスレジスタ 内の最終値	転送される ワード個数
(i)	$R_n$	$R_n - (WL * M)$	( 固定値 )
(ii)	$R_n$	$R_n$	$M$
(iii)	$R_n$	$R_n + (WL * M)$	$M$
(iv)	$R_n - (WL * M)$	$R_n$	$M$
(v)	$R_n - (WL * M)$	$R_n - (WL * M)$	$M$
(vi)	$R_n + (WL * M)$	$R_n$	( 固定値 )
(vii)	$R_n + (WL * M)$	$R_n + (WL * M)$	( 固定値 )

コプロセッサメモリアクセス命令は、以下のようなフラグを備えると有利である。

( i ) 前記開始アドレス値が、元々前記レジスタに記憶されていたアドレス値であるか、前記オフセットフィールドによって指定される変更されたアドレス値であるかを指定するフラグビット  $P$ 、

( i i ) 前記変更が、前記レジスタに元々記憶されていた値からオフセットフィールドによって指定される値を加算したものか減算したものであるかを指定するフラグビット  $U$ 、

( i i i ) 前記アドレスレジスタ内の前記最終値が、前記レジスタに記憶されな



おすべきか否かを指定するフラグビットW。

このようなフラグセットを使用して、コプロセッサは、高速で簡単な動作制御を行うことのできる構成となる。即ち、 $P \oplus E \oplus O \oplus R \oplus U$ を求めることによって、1個又はM個のデータワードが転送されるべきかを決定することができる。

更に、ベースレジスタがARMプログラムカウンタレジスタ (PC又はR1

5)であれば、転送されるワード数1を生成することができる。この場合、単一ワード転送を決定するロジックが、 $P \oplus E \oplus O \oplus R$  (U又は (ベースレジスタがPC))に変更される。

前述のコプロセッサと、中央処理装置及びメモリとの相互作用を制御する特性は多数の異なるフィールドに使用できる (例えば、浮動小数点コプロセッサ) 場合、データアクセスは、次のように、比較的規則的である。即ち、前記中央処理装置と前記コプロセッサがデジタル信号処理を行い、前記メモリと前記コプロセッサの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備える実施の形態に本発明は特に適している。

本発明を他の面から見ると、本発明は以下のステップを備えるデータ処理方法を提供する。即ち、

中央処理装置により、コプロセッサメモリアクセス命令を含む中央処理装置命令を実行してデータ処理操作を行うステップと、

前記中央処理装置に結合されたメモリにおいてデータワードを保持するステップと、

前記中央処理装置に結合されたコプロセッサにより処理される前記メモリ内のデータワードをアドレスするステップであって、前記メモリは、前記中央処理装置により実行されるコプロセッサメモリアクセス命令の制御下で、複数のアドレッシングモードの1つを使用するステップと

を備え、

前記コプロセッサメモリアクセス命令の少なくとも1つは、前記複数のアドレッシングモードのどれを中央処理装置が使用して前記メモリをアクセスするかを制御するアドレッシングモード情報を含み、

前記コプロセッサは、前記少なくとも1つのコプロセッサメモリアクセス命令に  
応答して、前記アドレッシングモード情報の少なくとも一部を使用して何個のデ  
ータワードが前記メモリと前記コプロセッサの間で転送されるかを制御すること  
を特徴とする方法である。

本発明の実施の形態について、以下に添付図面を参照しながら、例を示す。

図1は、デジタル信号処理装置のハイレベルの構成を示し、

図2は、コプロセッサの入力バッファとレジスタ構成を示し、

図3は、コプロセッサ内のデータパス ( d a t a p a t h ) を示し、

図4は、レジスタからハイ又はローのビットを読むためのマルチプレクシング  
回路を示し、

図5は、好ましい実施の形態におけるコプロセッサにより使用されるレジスタ  
・リマッピング・ロジックを示すブロック図であり、

図6は、図5に示されたレジスタ・リマッピング・ロジックを更に詳しく示し  
、

図7は、ブロック・フィルタ・アルゴリズムを示す表であり、

図8は、中央処理装置と、メモリと、コプロセッサアクセスメモリ命令を実行  
するためのコプロセッサとを備えたシステムを模式的に示し、

図9は、図8のシステムの動作に対応するフロー・ダイアグラムである。

以下の説明において、セクション1では、中央処理装置と、メモリと、高速デ  
ジタル信号処理能力を持つコプロセッサとを備えたシステムについて説明する。  
セクション2は、セクション1のシステムの変形について述べるもので、そこで  
は、コプロセッサメモリアクセス命令が、転送されるデータワードの個数のコプ  
ロセッサによる制御をより簡単にすべく変更されている。

#### 請求の範囲

1. データ処理装置であって、

コプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ  
処理動作を行う中央処理装置 ( 2 ) と、

前記中央処理装置に結合され、データワードを保持するメモリ (8) と、

前記中央処理装置と前記メモリに結合されたコプロセッサ (4) であって、コプロセッサにより処理されるメモリ内のデータワードをアドレスするのに、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、複数のアドレッシングモードの 1 つを使用することを特徴とするコプロセッサ (4) と

を備えるデータ処理装置であって、

少なくとも 1 つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリをアクセスするのに複数のアドレッシングモードのどれを使用するかを制御するアドレッシングモード情報 (P, U, W, M) を含み、

前記コプロセッサは、前記アドレッシングモード情報の少なくとも一部 (P, U, M) を使用して、前記少なくとも 1 つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御することを特徴とするデータ処理装置。

2. 請求項 1 に記載の装置であって、前記少なくとも 1 つのコプロセッサ・メモリアクセス命令が、アドレス値を持つ前記中央処理装置内のレジスタを参照し、前記アドレスモード情報がオフセットフィールド (M) を含み、そこでは、アクセスされるべき前記メモリ内の開始アドレスが、前記少なくとも 1 つのコプロセッサ・メモリアクセス命令の実行に際して、前記アドレス値と前記オフセット値の少なくとも 1 つから決まることを特徴とする装置。

3. 請求項 2 に記載の装置であって、前記アドレス値への前記変更が、最終アドレス値を生成し、それが前記レジスタに書き戻されることを特徴とする装置。

4. 請求項 2 と 3 のいずれかに記載の装置であって、前記オフセットフィールド (M) の少なくとも一部が、前記コプロセッサにより使用されて、前記メモリ

と前記コプロセッサとの間で転送されるデータの個数を制御することを特徴とする装置。

5. 請求項 4 に記載の装置であって、前記アドレッシングモード情報が 1 つ又は 2 つ以上のフラグ (P, U) を持ち、それが、前記複数のアドレッシングモードの

どれが使用されているかを制御し、且つ、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを定める際に、前記オフセットフィールドが前記コプロセッサによって使用されるべきかどうかを制御することを特徴とする装置。

6. 請求項5に記載の装置であって、前記メモリと前記コプロセッサの間で何個のデータワードが転送されるかを定める際に、前記オフセットフィールドが前記コプロセッサによって使用されない場合、固定数のデータワードが前記メモリと前記コプロセッサとの間で転送されることを特徴とする装置。

9. 請求項8に記載の装置であって、前記コプロセッサがP E O R Uを求めて、データワードを1個又はM個転送すべきかを決定することを特徴とする装置。

10. 請求項8に記載の装置であって、前記レジスタが前記中央処理装置のプログラムカウンタレジスタPCであり、前記コプロセッサが、転送されるデータワードが1個かM個かを定めるために、P E O R (U O R (レジスタはPC))を求めることを特徴とする装置。

11. 以上の請求項のいずれかに記載の装置であって、前記中央処理装置及び前記コプロセッサが、デジタル信号処理を行い、前記メモリと前記コプロセッサとの間で転送されるデータワードが、前記メモリに記憶された係数値のアレイ内からの係数値を備えることを特徴とする装置。

12. 請求項6及び請求項7乃至11のいずれかに記載の装置であって、前記固定数のデータワードが単一のデータワードを含む事を特徴とする装置。

13. データ処理方法であって、

中央処理装置によりコプロセッサ・メモリアクセス命令を含む中央処理装置命令を実行してデータ処理動作を行うステップと、

データワードを、前記中央処理装置に結合されたメモリで保持するステップと、

、

前記中央処理装置と前記メモリに結合されたコプロセッサによって、前記中央処理装置により実行される前記コプロセッサ・メモリアクセス命令の制御下で、

複数のアドレッシングモードの1つを使用することによって、処理されるメモリ内のデータワードをアドレスするステップと

を備えるデータ処理方法であって、

少なくとも1つのコプロセッサ・メモリアクセス命令が、前記中央処理装置が前記メモリをアクセスするのに複数のアドレッシングモードのどれを使用するかを制御するアドレッシングモード情報を含み、

前記コプロセッサは、前記アドレッシングモード情報の少なくとも一部を使用して、前記少なくとも1つのコプロセッサ・メモリアクセス命令に応答して、メモリとコプロセッサとの間で転送されるデータワードが何個であるかを制御することを特徴とするデータ処理方法。

## 【 国際調査報告 】

## INTERNATIONAL SEARCH REPORT

International Application No.

PCT/GB 98/00083

## A. CLASSIFICATION OF SUBJECT MATTER

IPC 6 G06F9/38 G06F9/312

According to International Patent Classification (IPC) or to both national classification and IPC

## B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

## C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	S. B. FURBER: "VLSI RISC Architecture and Organization" 1989, MARCEL DEKKER, INC., NEW YORK XP002061358 201970	1-3, 11-13
Y	* page 261, section 4.1.9: 'Coprocessor Data Transfer Instructions' *	4-8
Y	EP 0 703 529 A (NIPPON ELECTRIC CO) 27 March 1996 see claims 1-5	4-8
A	FR 2 637 708 A (NIPPON ELECTRIC CO) 13 April 1990 see claim 1	4

☐ Further documents are listed in the continuation of box C.☒ Patent family members are listed in annex.

## \* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance: the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance: the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"Z" document member of the same patent family

Date of the actual completion of the international search

3 April 1998

Date of mailing of the international search report

20/04/1998

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentaan 2  
NL - 2280 HV Rijswijk  
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,  
Fax: (+31-70) 340-3016

Authorized officer

Daskalakis, T

## INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No

PCT/GB 98/00083

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0703529 A	27-03-96	JP 2682469 B	26-11-97
		JP 8095780 A	12-04-96
FR 2637708 A	13-04-90	JP 2015327 C	02-02-96
		JP 2103630 A	16-04-90
		JP 7048179 B	24-05-95

---

【要約の続き】

合、転送されるワード数はデフォルトの 1 になる。